DOCUMENTAȚIE AGENȚI

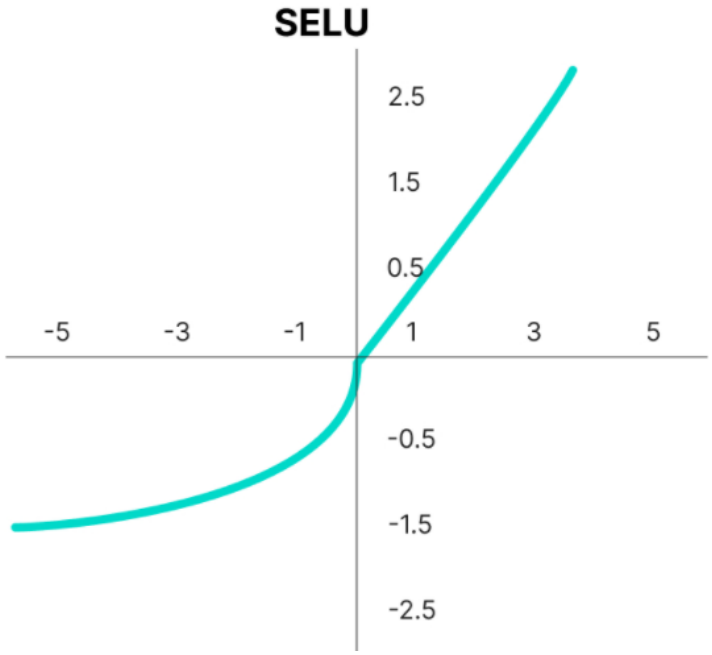
I. Funcții de activare

1. SELU

- ➔ Se ocupă de normalizarea internă
- ➔ Fiecare strat își pastreaza media si varianța
- ➔ folosește atât valori pozitive cât și negative ca să altereze media, ceea ce era imposibil în cazul ReLU deoarece nu poate returna valori negative
- ➔ E mai bun decat RELU deoarece ajută rețeaua să conveargă (normalizarea internă e și mai eficientă decât cea externă de la ReLU)
- ➔ Gradientul poate ajusta varianța
- ➔ Funcția de activare are nevoie de o regiune cu gradientul mai mare decât 1 pentru a o mări

$$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) & for\ x < 0 \\ x & for\ x \geqslant 0 \end{cases}$$

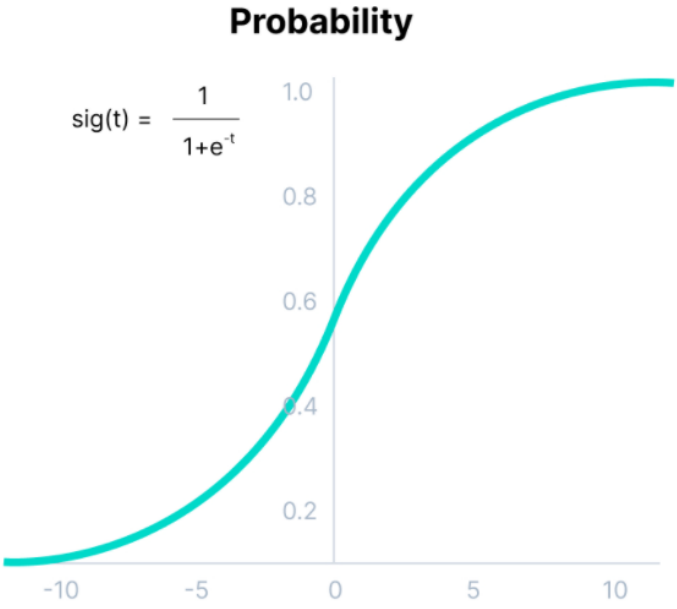
➢ Valorile pentru α și λ sunt predefinite



2. **Softmax**
   - ➔ Calculează probabilitățile relative
   - ➔ Folosită pentru clasificarea multiplă și de obicei pe ultimul strat

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

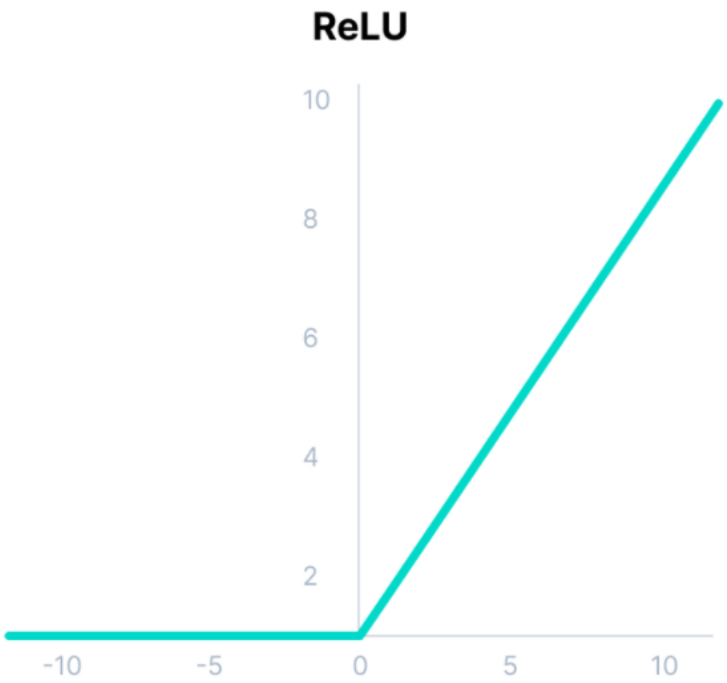
$\vec{z}$	Vectorul de input dat sub forma z0, z1, … , zk
z_i	Valorile vectorului de input
e^{z_i}	Funcția exponențială standard care e aplicată fiecărui element din vectorul de input. Aceasta returnează o valoare pozitivă., dar mică, în cazul în care inputul este negativ și crește cu atât mai mult cu cât crește și inputul. Nu este fixată în intervalul (0, 1) ceea ce este necesar în cazul unei probabilități
$\sum_{j=1}^{K} e^{z_j}$	Numitorul reprezintă termenul normalizat. Se asigură că suma tuturor valorilor de output este 1 și că acestea se află în intervalul (0, 1), constituind o distribuție de probabilitate validă
K	Numărul de clase ale clasificării multiple



3. **ReLU**
   - ➔ Permite propagarea
   - ➔ Nu activează toți neuronii în același timp, deci este eficientă
   - ➔ Neuronii vor fi deactivați dacă outputul transformării liniare e mai mic decât 0

$$ReLU$$

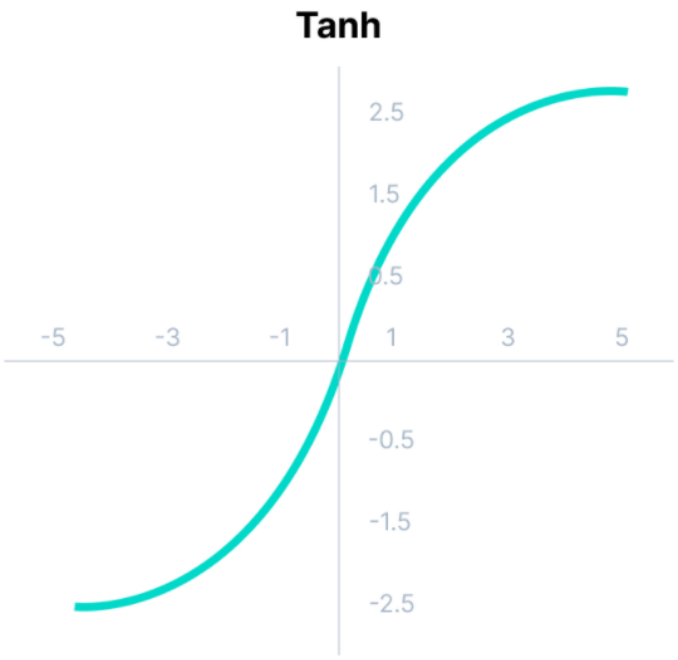
$$f(x) = max\ (0, x)$$



4. **Tanh**
   - ➔ De obicei e folosit în hidden_layer deoarece media acestuia va tinde la 0, astfel datele sunt centrate în origine și învățarea devine un proces mai ușor
   - ➔ Datorită acestei configurații, datele pot fi clasificate categoric ca “TRUE POSITIVE” sau “TRUE NEGATIVE”

$$Tanh$$

$$f(x) = \frac{\left(e^{x} - e^{-x}\right)}{\left(e^{x} + e^{-x}\right)}$$



II. Algoritmi de RL utilizați și cele mai bune rezultate

1. SARSA

➔ $Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$

➔ Actualizarea funcției Q(stare, acțiune) depinde de starea și acțiunea curentă a agentului, starea și acțiunea viitoare și recompensă

➔ Hiperparametri

- α: rata de învățare; determină rata cu care noile informații le vor suprascrie cele vechi
- γ: factor de reducere; determină importanța viitoarelor premii
- Condiția inițială: $Q(s_0, a_0)$; fiind un algoritm iterativ, se asumă o condiție inițială înainte de realizarea primei actualizări. O valoare inițială mică (optimistă) încurajează învățarea

2. Expected SARSA

➔ $Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$

➔ Ia suma ponderată a tuturor acțiunilor în raport cu probabilitatea de a executa acea acțiune

➔ Variație a lui SARSA care exploatează cunoștințele despre stocasticitate pentru a face actualizări cu varianță mai mică, îmbunătățind politica agentului

➔ Varianța este 0, ceea ce permite o rată de învățare de 1

➔ Converge sub aceleași condiții ca și SARSA

3. Q-Learning

➔ $Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_{t+1} + \gamma max_a Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$

➔ Învață valoarea politicii optime independent de acțiunea agentului

➔ Utilizează diferențe temporale pentru a estima valoarea așteptatăș diferența temporală este un agent de învățare prin episoade al unui mediu, fără să aibă cunoștințe anterioare

4. DQN

➔ Aproximează valoarea funcției stare-acțiune: Q(stare, acțiune, parametri)

➔ Avansarea către următoarele stări: Q(stare, acțiune) ar trebui să se apropie de:

- targetQ = (recompensă + Q(stareUrmătoare, următoareaCeaMaiBeneficăAcțiune, parametri)
- targetQ = recompensă dacă nu mai există alte viitoare stări

➔ Funcția de pierdere: eroarea medie (pătratică) sau funcția de pierdere Huber (folosită în regresia robustă, este mai sensibilă la valorile care nu se încadrează în grafic decât eroarea medie)

- Aplicată între Q(stare, acțiune) și targetQ

➔ Calcularea gradientului parametrilor în raport cu funcția de pierdere ajută la actualizarea lor pentru a se centra în jurul unei valori comune

III. Rezultate

1. DQN BASE

a.

```
self.hidden_layers.append(tf.keras.layers.Dense(128, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(64, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
200	38

b.

```
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='linear'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
1400	81

c.

```
self.hidden_layers.append(tf.keras.layers.Dense(64, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
300	39.3

d.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
300	31.2

e.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
200	-1.5

f.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
3500	150

g.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
110	119.199
120	193.5
180	141.7
260	122.2
420	114.5
440	121.09
560	120.2
770	176.0
960	168.9
1000	175.19

1030	294.29
1070	241.6
1600	237.3
2080	225.3
2250	229.8

2. DNQ TARGET NETWORK

a.

```
 self.hidden_layers.append(tf.keras.layers.Dense(32, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
850	115.46
1500	131.199
1920	143.5

b.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
900	141.44
1650	151.6

c.

```
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='softmax'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='softmax'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='softmax'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
1000	86

d.

```
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='softmax'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
1000	102.10

e.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='selu'))
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layersa.append(tf.keras.layers.Dense(32, activation='selu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
1500	130.04

f.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='selu'))
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='selu'))
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
1000	142.6

g.

```
 self.hidden_layers.append(tf.keras.layers.Dense(32, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='selu'))
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
400	163.26

3. DQN TARGET NETWORK AND EXPERIENCE REPLAY

a.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='softmax'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
1000	86.9

b.

```
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='softmax'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
500	31.7

c.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='softmax'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
1000	82.2

d.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='selu'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='selu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
2700	176.5

e.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
300	73.8

4. DQN ALL

a.

```
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='softmax'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='softmax'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='softmax'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
1000	86

b.

```
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='softmax'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
1000	102.10

c.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(8, activation='tanh'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
1000	102.10

d.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='selu'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='selu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
790	202.1
890	135.69

e.

```
self.hidden_layers.append(tf.keras.layers.Dense(32, activation='tanh'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
self.hidden_layers.append(tf.keras.layers.Dense(16, activation='relu'))
```

NUMBER OF EPISODES	AVERAGE REWARDS
500	252.6
860	140

IV. Concluzii

- Combinația funcțiilor de activare ReLU și Tanh a avut cea mai mare eficiență la DQN_BASE
- Funcția de activare SELU a avut cea mai mare eficiență la DQN_TARGET_NETWORK_AND_EXPERIENCE_REPLAY
- Cu cât creșteam numărul nodurilor sau al straturilor, cu atât făcea mai multe mutări ilegale și executarea devenea din ce în ce mai greoaie