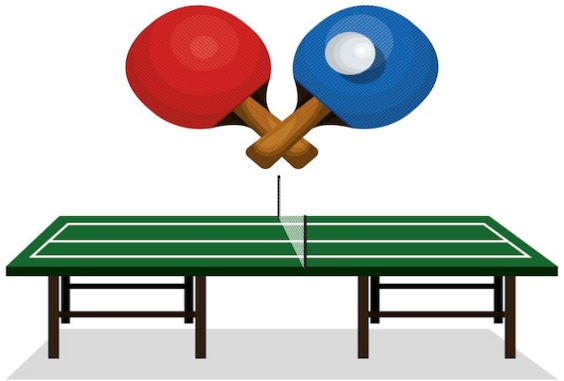
GRUPPO 22

«Machine Learning»

Tennis Table Tournament



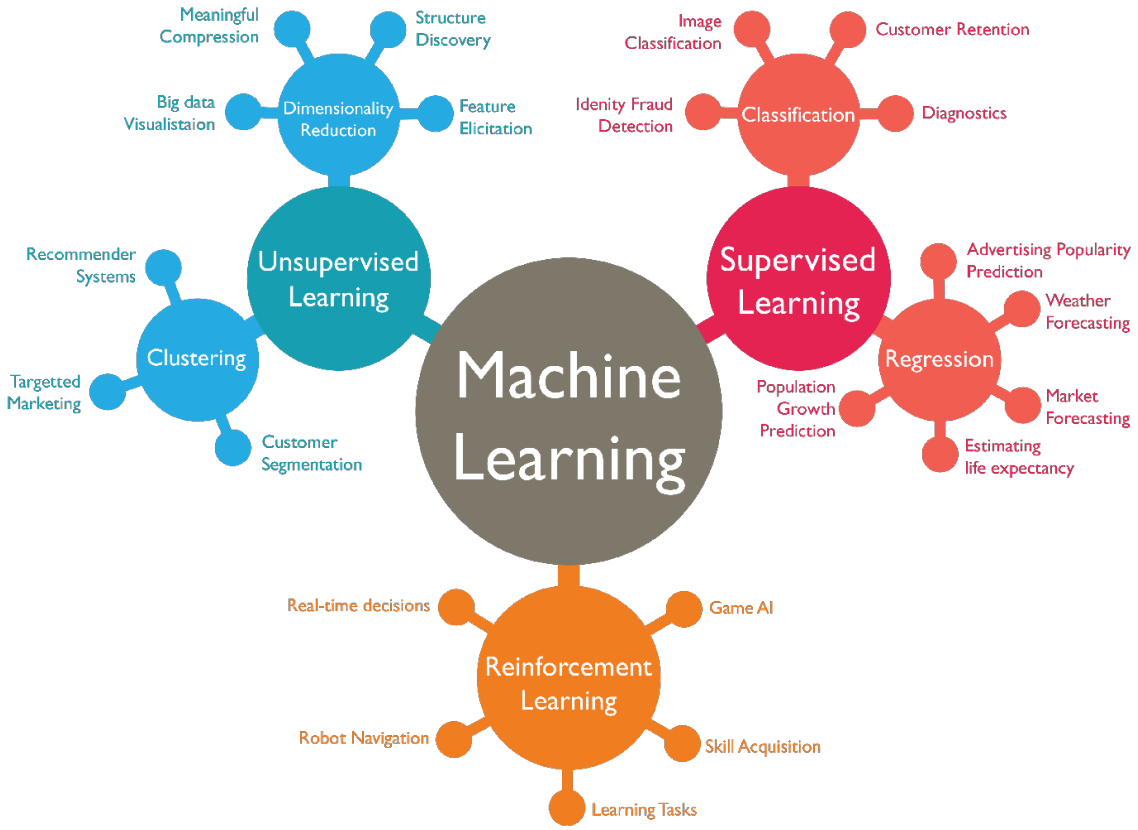
MEMBRI:

Antony Storti
Paola Saggiomo
Miriam Valentino
Damiano Noschese

DIEM
Dipartimento di
Ingegneria dell'Informazione ed Elettrica
e Matematica applicata

MUR
DIPARTIMENTO
DI ECCELLENZA
2023-2027

LA SOLUZIONE PROPOSTA:



Supervised Learning + RL

- **Supervised** → Imparare a inseguire la pallina usando la *cinematica inversa*.
- **RL** → Insegnare al braccio robotico come usare correttamente la racchetta.

Supervised Learning: «*Scelta delle feature*»

```
============================================================
State list
Index       Description
0-10        Current joint positions
11-13       Paddle center position (x,y,z)
14-16       Paddle normal versos (x,y,z)
17-19       Current ball position (x,y,z)
20-22       Current ball velocity (x,y,z)
23-25       Opponent paddle center position (x,y,z)
26          Game waiting, cannot move (0=no, 1=yes)
27          Game waiting for opponent service (0=no, 1=yes)
28          Game playing (i.e., not waiting) (0=no, 1=yes)
29          Ball in your half-field (0=no, 1=yes)
30          Ball already touched your side of table (0=no, 1=yes)
31          Ball already touched your rob (0=noto, 1=yes)
32          Ball in opponent half-field (0=no, 1=yes)
33          Ball already touched opponent side of table (0=no, 1=yes)
34          Your score
35          Opponent score
36          Simulation time

============================================================
```

Per insegnare al braccio robotico i movimenti da compiere per inseguire correttamente la pallina, usiamo la **Regressione Multivariata (*Multipla*)** :

- Valori da predire → **11 JOINTS**
- Predittori → **Ball Position**

Abbiamo dovuto raccogliere una grande quantità di dati per addestrare questa rete ed evitare problemi di *overfitting*:

- **Train Set** → 980.000 stati
- **Validation Set** → 250.000 stati
- **Test Set** → 125.000 stati

.DIEM
Dipartimento di
Ingegneria dell'Informazione ed Elettrica
e Matematica applicata
MUR DIPARTIMENTO DI ECCELLENZA 2023-2027

Supervised Learning:

Modello di Regressione Multivariata

Il modello di regressione multivariata può essere rappresentato come:

$$\begin{cases} Y1 = \beta_{10} + \beta_{11}X1 + \beta_{12}X2 + \beta_{13}X3 + \epsilon_1 \\ Y2 = \beta_{20} + \beta_{21}X1 + \beta_{22}X2 + \beta_{23}X3 + \epsilon_2 \\ \vdots \\ Y11 = \beta_{110} + \beta_{111}X1 + \beta_{112}X2 + \beta_{113}X3 + \epsilon_{11} \end{cases}$$

Dove:

- $Y1, Y2, ..., Y11$ sono le variabili dipendenti.
- $X1, X2, X3$ sono le variabili indipendenti.
- β sono i coefficienti di regressione.
- ϵ è l'errore residuo.

Supervised Learning:

Funzione di Costo

La funzione di costo utilizzata per la regressione multivariata è tipicamente l'errore quadratico medio (MSE):

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{Y}_i - \hat{\mathbf{Y}}_i\|^2$$

Dove:

- n è il numero di campioni.
- $\mathbf{Y}_i$ è il vettore delle variabili dipendenti osservate per il campione i.
- $\hat{\mathbf{Y}}_i$ è il vettore delle variabili dipendenti predette per il campione i.

Algoritmo di Ottimizzazione

Utilizziamo l'algoritmo di discesa del gradiente per aggiornare i pesi e i bias:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \text{MSE}}{\partial \mathbf{W}}$$
$$\mathbf{b} \leftarrow \mathbf{b} - \eta \frac{\partial \text{MSE}}{\partial \mathbf{b}}$$

Dove:

- η è il tasso di apprendimento.
- $\frac{\partial \text{MSE}}{\partial \mathbf{W}}$ e $\frac{\partial \text{MSE}}{\partial \mathbf{b}}$ sono i gradienti della funzione di costo rispetto ai pesi e ai bias.

Supervised Learning:

Rappresentazione Complessiva

Mettiamo insieme tutto:

1. **Input**: $\mathbf{X} = [X1, X2, X3]$
2. **Strato Nascosto**: $\mathbf{Z} = \text{ReLU}(\mathbf{W}^{(1)}\mathbf{X} + \mathbf{b}^{(1)})$
3. **Output**: $\mathbf{Y} = \mathbf{W}^{(2)}\mathbf{Z} + \mathbf{b}^{(2)}$

Quindi, il modello matematico della rete neurale MLP per la regressione multivariata con un solo strato nascosto è:

$$\mathbf{Y} = \mathbf{W}^{(2)} \left(\text{ReLU} \left(\mathbf{W}^{(1)}\mathbf{X} + \mathbf{b}^{(1)} \right) \right) + \mathbf{b}^{(2)}$$

Dove:

1. $\mathbf{X}$: Matrice degli input
2. $\mathbf{W}^{(1)}$: Matrice dei pesi per il primo strato nascosto
3. $\mathbf{b}^{(1)}$: Vettore dei bias per il primo strato nascosto
4. $\mathbf{Z}$: Matrice delle attivazioni del primo strato nascosto
5. $\mathbf{W}^{(2)}$: Matrice dei pesi per lo strato di output
6. $\mathbf{b}^{(2)}$: Vettore dei bias per lo strato di output
7. $\mathbf{Y}$: Matrice degli output

La funzione ReLU è definita per un dato input x come:

$$\text{ReLU}(x) = \max(0, x)$$

Questo significa che per qualsiasi valore di x:

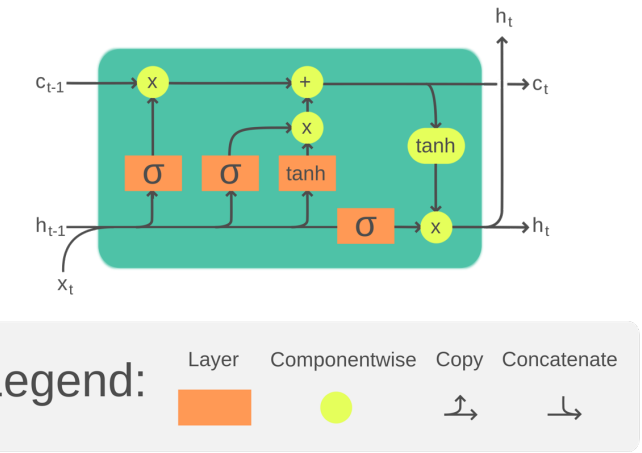
- Se $x > 0$, allora $\text{ReLU}(x) = x$.
- Se $x \leq 0$, allora $\text{ReLU}(x) = 0$.

In altre parole, la funzione ReLU "attiva" il neurone se l'input è positivo (ritornando l'input stesso), mentre se l'input è negativo o zero, il neurone rimane inattivo (ritornando zero). Questo comportamento aiuta a introdurre non-linearità nelle reti neurali, rendendole capaci di apprendere modelli più complessi e di migliorare la convergenza durante l'addestramento.

Supervised Learning: «MLP vs LSTM»

Le reti LSTM, acronimo di Long Short-Term Memory, rappresentano una particolare architettura di reti neurali ricorrenti (RNN) progettata per gestire problemi in cui è necessario tenere conto delle dipendenze a lungo termine all'interno di una sequenza di dati. Ecco alcuni punti chiave riguardanti le LSTM:

1. **Cella di Memoria**: Le LSTM utilizzano una cella di memoria che permette di mantenere e aggiornare informazioni nel corso del tempo, consentendo di catturare dipendenze a lungo termine.
2. **Porte**: Le LSTM sono dotate di tre porte fondamentali per gestire il flusso di informazioni:
   - **Porta di Dimenticanza (Forget Gate)**: Decide quali informazioni mantenere o dimenticare dalla cella di memoria.
   - **Porta di Ingresso (Input Gate)**: Decide quali nuove informazioni aggiornare nella cella di memoria.
   - **Porta di Uscita (Output Gate)**: Decide quale parte della cella di memoria sarà utilizzata per la predizione o l'output del modello.
3. **Funzioni di Attivazione Non Lineari**: Le LSTM utilizzano funzioni di attivazione non lineari, come la sigmoide e la tangente iperbolica, per controllare il flusso di informazioni attraverso le porte.



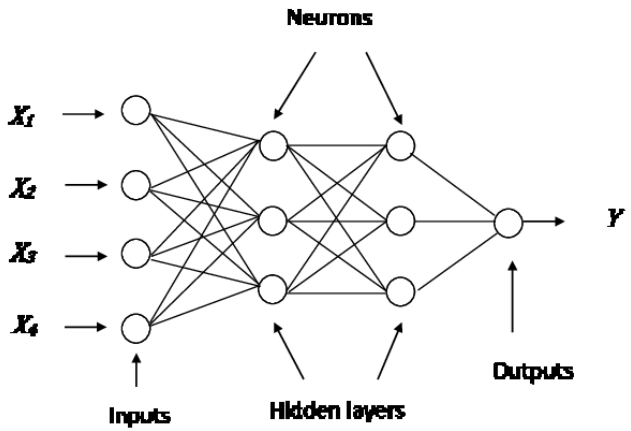
- Nonostante in prima battura questa sembrasse la rete più adatta al problema, abbiamo notato sperimentalmente che **le prestazioni tra essa e la più semplice rete MLP non differivano sensibilmente.**
- Per tale motivo, abbiamo optato per l'uso di una **rete MLP**, in quanto, computazionalmente **meno esosa di risorse**!

DIEM
Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica applicata
MUR DIPARTIMENTO DI ECCELLENZA 2023-2027

Supervised Learning: «Implementazione»

```python
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size1, output_size, dropout_prob=0.5):
        super(MLP, self).__init__()
        self.hidden1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.dropout = nn.Dropout(p=dropout_prob)
        self.output = nn.Linear(hidden_size1, output_size)

    def forward(self, x):
        x = self.hidden1(x)
        x = self.relu1(x)
        x = self.dropout(x)
        x = self.output(x)
        return x
```



Benefici dell'Early Stopping:

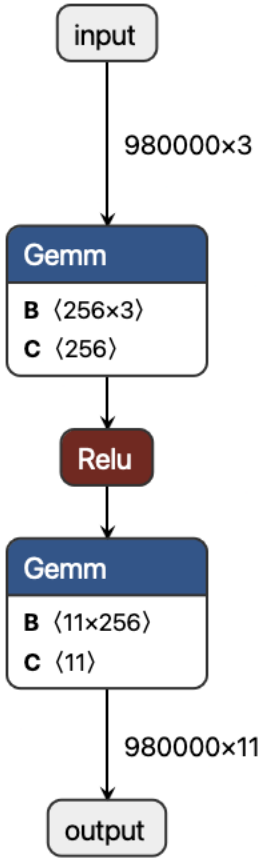
- **Prevenzione dell'overfitting:** Interrompe il training del modello prima che si adatti troppo ai dati di training, migliorando la capacità di generalizzazione del modello.
- **Risparmio di risorse:** Riduce il tempo e le risorse necessarie per il training, poiché il training viene interrotto non appena il modello smette di migliorare sul set di validazione.

Benefici del Dropout:

- **Prevenzione dell'overfitting:** Disattivando casualmente i neuroni, il modello non può fare affidamento eccessivo su particolari neuroni o connessioni, migliorando la sua capacità di generalizzazione.
- **Miglioramento delle prestazioni:** Il dropout può migliorare le prestazioni del modello su dati di test, riducendo il divario tra l'errore di training e l'errore di test.

Supervised Learning: «Tuning della rete»

```
Test MSE: 0.017468770796133817
```




```
# RISULTATI:

# 128/50 --> 5 prese --> Margine +- 3%

# 200/50 --> 8 prese --> Margine +- 3%

# 256/50 --> 8 prese --> Margine +- 3% --> OK
# 256/75 --> 7 prese --> Margine +- 3%

# 300/50 --> 5 prese --> Margine +- 3%
# 350/50 --> 8 prese --> Margine +- 3%

# 450/50 --> 5 prese --> Margine +- 3%

# 512/25 --> 7 prese --> Margine +- 3%
# 512/50 --> 7 prese --> Margine +- 3%
```

```
Predictions: tensor([[-2.8679e-01,  9.9721e-02,  3.1413e+00,  1.2754e-01, -1.6848e-03,
          7.9046e-01, -3.5551e-03,  7.4893e-01,  1.2465e-03,  8.3805e-01,
          1.5443e+00]])
Expected values: tensor([[-3.0000e-01,  8.2765e-03,  3.1416e+00,  8.2689e-02,  9.7625e-03,
          7.6662e-01,  1.4106e-03,  7.2403e-01,  3.0737e-04,  9.2572e-01,
          1.5717e+00]])
Difference: tensor([[ 0.0132,  0.0914, -0.0003,  0.0449, -0.0114,  0.0238, -0.0050,  0.0249,
          0.0009, -0.0877, -0.0274]])
```


.DIEM
Dipartimento di
Ingegneria dell'Informazione ed Elettrica
e Matematica applicata
MUR
DIPARTIMENTO DI ECCELLENZA
2023-2027

Reinforcement Learning: «Strategia»

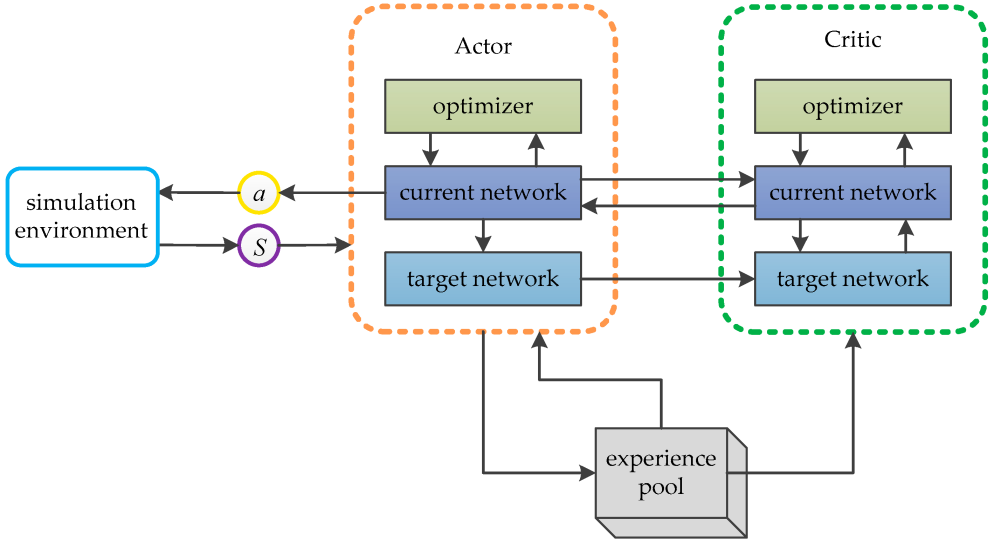
Index	Type	Values	Description
0	Translation	-0.3 ... 0.3	Forward-Backward Slider. Positive Values are forward.
1	Translation	-0.8 ... 0.8	Left-Right Slider. Positive Values are to the right.
2	Rotation	Any	Rotation around the vertical axis (Z).
3	Rotation	-π/2 ... π/2	Pitch of the first arm link.
4	Rotation	Any	Roll of the first arm link.
5	Rotation	-π*3/4 ... π*3/4	Pitch of the second arm link.
6	Rotation	Any	Roll of the second arm link.
7	Rotation	-π*3/4 ... π*3/4	Pitch of the third arm link.
8	Rotation	Any	Roll of the third arm link.
9	Rotation	-π*3/4 ... π*3/4	Pitch of the pad.
10	Rotation	Any	Roll of the pad.

```
# Gestisco gli ultimi due giunti (quelli del Paddle)
# MOTIVO --> Sono gli unici che vengono gestiti dal RL
action = actor_model(state_tensor).detach().numpy().flatten() + noise.noise()
action[0] = np.clip(action[0], -pi * 3 / 4, pi * 3 / 4)  # Clip delle azioni del Giunto 9 --> +- 45°

# Invio dei giunti --> (Da 0 a 8): MPL | (Da 9 a 10): DDPG
joints = [state[0], state[1], state[2], state[3], state[4], state[5], state[6], state[7], state[8], action[0],
          action[1]]
cli.send_joints(joints)
```

- Il **MLP** permette al braccio di seguire la pallina:
  - Lo fa muovendo i **JOINTS [0:8]**
- Useremo il **RL** per permette al braccio di eseguire un lancio corretto:
  - Muoverà solo i **JOINTS [9:10]**

Reinforcement Learning: «DDPG»



Proprio come il metodo Attore-Critico, abbiamo due reti:

1. Attore - Propone un'azione dato uno stato.
2. Critico: prevede se l'azione è buona (valore positivo) o cattiva (valore negativo) dato uno stato e un'azione.

DDPG utilizza altre due tecniche non presenti nel DQN originale:

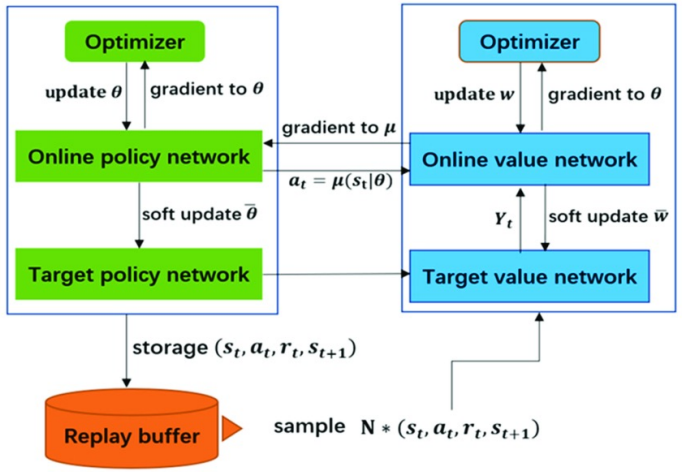
Innanzitutto, utilizza due reti Target.

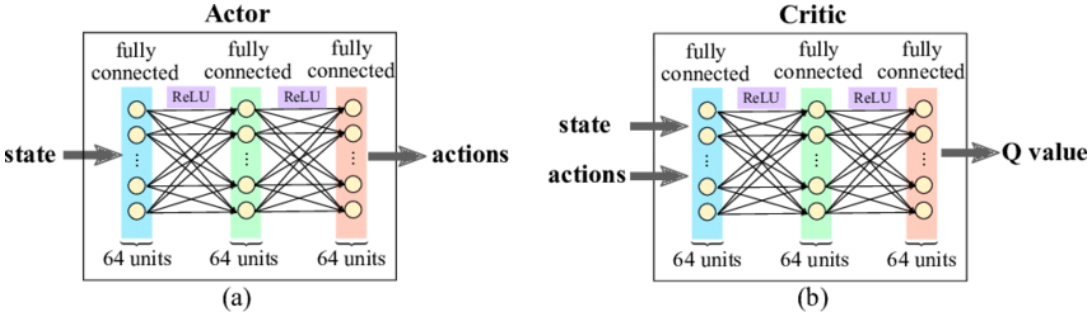
Perché? Perché aggiunge stabilità all'allenamento. In breve, stiamo imparando dalle stime gli obiettivi e le reti target vengono aggiornati lentamente, mantenendo quindi i nostri obiettivi stimati stabile.

Concettualmente è come dire: "Ho un'idea su come suonare così bene, Lo proverò per un po' finché non troverò qualcosa di meglio", invece di dire "Imparerò di nuovo a giocare a questo gioco dopo ogni partita mossa". Vedi questa risposta StackOverflow .

In secondo luogo, utilizza Experience Replay.

Memorizziamo l'elenco di tuple `(state, action, reward, next_state)`, e invece di imparando solo dall'esperienza recente, impariamo campionando tutta la nostra esperienza accumulato finora.

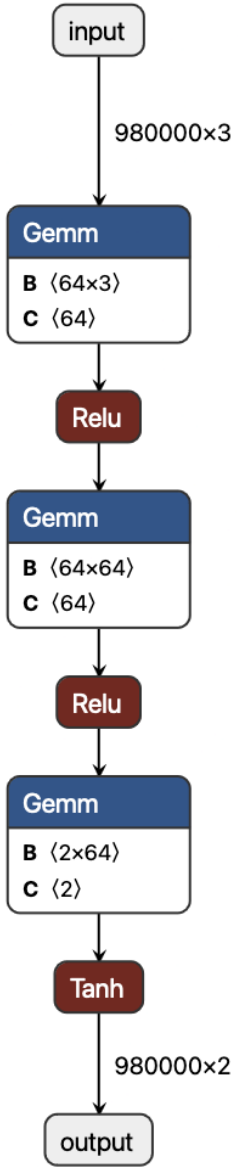




Reinforcement Learning: «Implementazione»

```python
class ActorNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(ActorNetwork, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, output_size)
        self.relu = nn.ReLU()
        self.tanh = nn.Tanh()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.tanh(self.fc3(x))
        return x
```



Attore (Actor):

- L'attore è responsabile di apprendere e mantenere la politica ottimale $\mu(s)$, che mappa gli stati s direttamente alle azioni a.
- L'obiettivo dell'attore è massimizzare il valore Q predetto dal critico, cioè scegliere azioni che portino a un alto valore Q.
- Matematicamente, l'attore mira a risolvere il seguente problema di ottimizzazione:

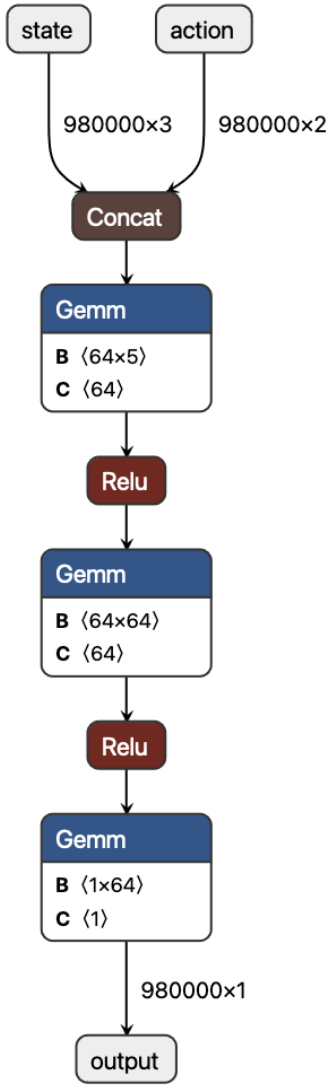
$$\max_{\theta_\mu} \mathbb{E}_{s\sim\rho^\beta}[Q(s,\mu(s))]$$

dove θ_μ sono i parametri dell'attore, ρ^β è la distribuzione degli stati sotto la politica attuale μ, e $Q(s,\mu(s))$ è il valore Q predetto dal critico per l'azione scelta dall'attore.

Reinforcement Learning: «Implementazione»

```python
class CriticNetwork(nn.Module):
    def __init__(self, state_size, hidden_size, action_size):
        super(CriticNetwork, self).__init__()
        self.state_size = state_size
        self.action_size = action_size
        self.fc1 = nn.Linear(state_size + action_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, out_features: 1)
        self.relu = nn.ReLU()

    def forward(self, state, action):
        x = torch.cat( tensors: [state, action], dim=1)
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```



Critic (Critic):

- Il critico stima la funzione valore Q, cioè valuta la qualità di una coppia stato-azione (s, a).
- Il compito principale del critico è apprendere una funzione $Q(s, a)$ che approssimi il ritorno atteso partendo dallo stato s e scegliendo l'azione a, seguito dalla politica ottimale.
- Durante l'addestramento, il critico viene aggiornato per minimizzare l'errore di differenza temporale (TD) tra i valori Q predetti e i valori Q target, calcolati usando una versione target della rete critica.
- Matematicamente, il critico mira a minimizzare la seguente funzione obiettivo:

$$\mathcal{L}(\theta_Q) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[(r + \gamma Q'(s', \mu'(s'; \theta_{\mu'}); \theta_{Q'}) - Q(s, a; \theta_Q))^2]$$

dove θ_Q sono i parametri del critico, $\mathcal{D}$ è l'insieme di esperienze campionate dall'ambiente, γ è il fattore di sconto, $Q'(s', \mu'(s'; \theta_{\mu'}); \theta_{Q'})$ è il valore Q target calcolato usando le reti target, e $Q(s, a; \theta_Q)$ è il valore Q predetto dal critico.

DIEM
Dipartimento di
Ingegneria dell'Informazione ed Elettrica
e Matematica applicata
MUR
DIPARTIMENTO DI ECCELLENZA
2023-2027

GRAZIE PER L'ATTENZIONE