UNIVERSIDAD MAYOR REAL Y PONTIFICIA DE SAN FRANCISCO XAVIER DE CHUQUISACA

FACULTAD DE CIENCIAS Y TECNOLOGIA

FUERZA MAXIMA EN HIP THRUST TREN INFERIOR MEDIANTE APRENDISAJE POR REFUERZO

MATERIA: SIS 420 INTELIGENCIA ARTIFICIAL I

DOCENTE: ING. PACHECO LORA CARLOS WALTER

UNIVERSITARIA: GOMEZ GARCIA FLORES DAFNE JULY

FECHA: 28/11/2025

SUCRE – BOLIVIA

FUERZA MAXIMA EN HIP THRUST TREN INFERIOR

MEDIANTE APRENDISAJE POR REFUERZO

El enfoque de un gimnasio (GYM) es un ambiente cerrado donde se realizan ejercicios físicos, equipado con máquinas, pesas y otros implementos para mejorar la salud y la condición física.
En este proyecto, se desarrolla un entorno de simulación inteligente basado en aprendizaje por refuerzo para optimizar la fuerza (1RM) en el ejercicio Hip Thrust. El entrenamiento se enfoca en un solo músculo del tren inferior, entrenamiento de fuerza es un proceso complejo que depende de múltiples factores, como la carga, las repeticiones y la fatiga.
Mediante el algoritmo Q-Learning, el sistema aprende de manera autónoma cuál es la mejor decisión de entrenamiento en cada condición del deportista, simulando el comportamiento de un entrenador inteligente.

OBJETIVO GENERAL:

Desarrollar un Sistema de Aprendizaje por Refuerzo que permita optimizar progresivamente la Fuerza Máxima (1RM) en el ejercicio de Hip Thrust del tren inferior. El sistema debe ser capaz de gestionar la carga de entrenamiento para evitar el sobre entrenamiento y garantizar la obtención de resultados óptimos para el deportista.

OBJETIVO ESPECIFICOS:

Modelar el entrenamiento como entorno de aprendizaje por refuerzo.
Definir estados en función de carga, repeticiones y fatiga.
Implementar el algoritmo Q-Learning.
Visualizar el progreso de la fuerza máxima mediante gráficos.
Mostrar una bitácora realista de sesiones de entrenamiento.

FUNDAMENTO TEORICO:

Aprendizaje por Refuerzo es una rama de Machine Learning donde un agente aprende interactuando con un entorno mediante prueba y error. El agente recibe recompensas o penalizaciones dependiendo de sus decisiones, con el objetivo de maximizar la recompensa acumulada.

ALGORITMO Q-LEARNING:

El **Q-Learning** es un algoritmo de aprendizaje por refuerzo sin modelo, que aprende la utilidad de cada acción en cada estado usando la ecuación de Bellman.

$$Q(s,a)=Q(s,a)+\alpha[r+\gamma\max(Q(s',a'))-Q(s,a)]$$

Donde:
α = tasa de aprendizaje
γ = factor de descuento
r = recompensa

MODELADO DEL ENTORNO DE ENTRENAMIENTO:

El ejercicio de fuerza máxima en Hip Thrust del tren inferior mediante Aprendizaje por Refuerzo es modelado como un entorno con:

- ✓ **Estados**

Variable	**Niveles**	**Descripción físico**	**Razón de Inclusión**
S1: Peso relativo	0,1,2,3	Nivel de carga en relación con la fuerza máxima 1RM conocida del usuario. Mapeo físico: Bajo 80% es una carga de calentamiento, Medio 85% si domina fácilmente, Alto 90% domina pero el esfuerzo es significativo, Máximo 95% peso retar que se combina con fatiga alta, pero tiene una alta probabilidad de causar fallo y recompensa.	Indica la intensidad de la sesión.
S2: Repeticiones	0,1,2	Rango, volumen de repeticiones Fuerza Máxima 5 R. carga muy pesada que recluta la mayor cantidad de fibra muscular, Hipertrofia Crecimiento 10 R. razón más común para ganar musculo, Resistencia/Volumen 15 R. más ligero enfocado en volumen total y la resistencia a la fatiga.	Indica el volumen del estímulo y el objetivo de la sesión(fuerza vs hipertrofia)
S3: Fatiga	0,1,2	Nivel de recuperación del musculo Baja (musculo totalmente descansado), Media (el musculo esta algo cansado de la última sesión y Alta (musculo cansado o sobreentrenado).	Factor crítico que determina si el usuario puede progresar o fallara.

Total, de estados: 4 x 3 x 3 = 36 estados

- ✓ **Acciones**

El agente puede elegir 5 acciones:

Índice	**Acción A**	**Impacto de decisión**	**Resultado esperado del agente**

0	Mantener	Repetir la carga y repeticiones de la última sesión.	Asegura la recompensa de mantenimiento **(+3)**
1	Subir Repeticiones	Aumentar el nivel de repeticiones **(S2)**	Aumentar el estímulo de forma segura sin subir el riesgo de fallo.
2	Subir Carga	Aumentar el nivel de peso relativo **(S1)**	Buscar la sobrecarga progresiva, forma más rápida de obtener **(+10)**
3	Bajar Carga	Disminuir el peso relativo **(S1)**	Minimizar el riesgo de fallo su penalidad de **(-5)** cuando la fatiga es alta.
4	Subir ambos	Aumentar **(S1)** y **(S2)** simultáneamente.	La acción más agresiva con alto riesgo y alta recompensa potencial.

Función de transición:

1. Aplica acción: la acción elegida (A) determina el nuevo estado de peso y repeticiones (S1 y S2)
2. Calcula riesgo: usa la fatiga(S3) y el peso planificado para calcular la probabilidad de fallo.
3. Determina el resultado: si hay fallo probabilidad alta, el entrenamiento fracasa, si hay éxito si se actualiza la fuerza máxima (Fmax) del usuario usando la constante ganancia K.
4. Actualiza el estado: genera el estado siguiente (S') la fatiga se incrementa si la sesión fue intensa o falla y se reduce si fue fácil.

- ✓ **Recompensas**

Resultado de la sesión	**Recompensa**	**Razón (Lógica Programada)**
Sobrecarga optima	+10	Se logra el progreso de la fuerza (Fmax) el objetivo principal del agente.
Mantenimiento	+3	La sesión fue exitosa, pero no lo suficientemente intensa para causar progreso.
Fallo	-5	El peso fue demasiado alto y la fatiga era extrema, penalidad baja incrementando el riesgo.
Baja Intensidad	-1	Se eligió una carga demasiado fácil, penalidad pequeña por perder el tiempo.

Nuevo ValorQHipThrust=(1-α)* Q(s,a)+ α* (Ganancia de puntaje/perdida por fallo +γmaxa'(Q(Estado Futuro,a'))

FUNCIONAMIENTO DEL SISTEMA:

1. Se inicializa la fuerza máxima (F_max).
2. El agente selecciona una acción usando épsilon-greedy.
3. Se calcula el estímulo del entrenamiento.
4. Se determina si hay progreso o fallo.

5. Se actualiza la Q-Table.
6. Se actualiza el estado.
7. Se repite por miles de episodios.

ENTRENAMIENTO DEL AGENTE:

1. Episodios: 10.000 (días de entrenamiento simulado)
2. Alpha α: 0.1
3. Gamma γ: 0.9
4. Épsilon inicial: 1.0
5. Épsilon mínima: 0.01

Durante el entrenamiento, el agente pasa de explorar aleatoriamente a explorar el conocimiento aprendido.

POLITICA OPTIMA:

Una vez entrenado el agente, se analiza la Q-Table para obtener la mejor acción en cada estado, esto representa la estrategia optima de entrenamiento

BITACORA DE ENTRENAMIENTO:

Se ejecuta una simulación de días donde:

1. Se muestra la fatiga diaria
2. El peso planificado
3. Las repeticiones
4. La recompensa obtenida
5. El cambio en la fuerza máxima

Esto demuestra la credibilidad practica del modelo.

VISUALIZACION DE RESULTADOS:

Se genera un gráfico que muestra:

1. La evolución del 1RM por episodio.
2. Una línea de referencia con la fuerza inicial.
3. El crecimiento progresivo de la fuerza.

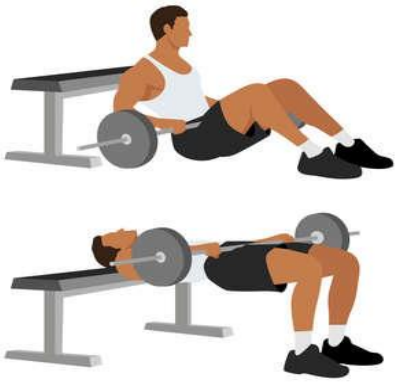
Esto permite validar visualmente el aprendizaje del agente.

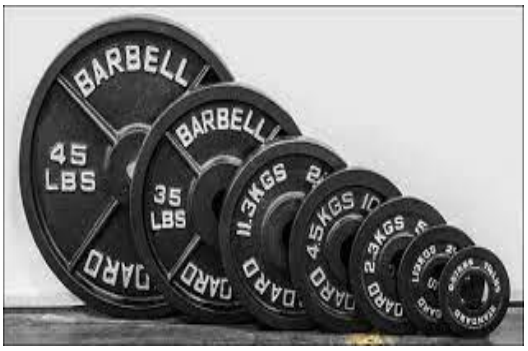
RESULTADOS:

1. El agente logra incrementar la fuerza máxima de manera progresiva
2. Evita entrenamientos excesivos mediante penalizaciones.
3. Aprende a priorizar acciones que generan mayor estimulo efectivo.
4. Simula un comportamiento realista de planificación de entrenamiento.

CONCLUSIONES:

El aprendizaje por refuerzo es aplicable al ámbito del entrenamiento físico, ya que el sistema actúa como un entrenador virtual inteligente que el modelo permite experimentar sin riesgo físico y puede extenderse a otros ejercicios.





CODIGO FUENTE:

FUERZA MAXIMA EN HIP THRUST TREN INFERIOR MEDIANTE APRENDISAJE POR REFUERZO

IMPORTAR LIBRERIAS

```
import numpy as np
import random
from collections import defaultdict
import matplotlib.pyplot as plt
```

PASO I EL ENTORNO SIMULADOR (HIP THRUST)
ENTORNO DE SOBRECARGA PROGRESIVA SIMULANDO EL ENTRENAMIENTO DE HIP THRUST TREN INFERIOR, EL OBJETIVO ES AUMENTAR LA FUERZA BASE (F_max) DEL USUARIO A LARGO PLAZO.

```
class HipThrustTrenInferior:

    def __init__(self, initial_fmax=100.0, k_gain=0.005):

        # PARAMETRO DE ENTORNO

        self.k_gain = k_gain
        self.initial_fmax = initial_fmax
        self.F_max = initial_fmax

        # ESPACIOS DE ESTADO

        self.state_space = (4, 3, 3)

        self.observation_space_size = np.prod(self.state_space)


        # ESPACIO DE ACCION

        self.action_space_size = 5

        self.current_state = None

        self.reset()
```

```
    def reset(self):
        # REINICIA EL ESTADO INICIAL
        self.current_state = (0, 1, 0)
        self.F_max = self.initial_fmax
        return self._state_to_index(self.current_state)

    def _state_to_index(self, state):
        # CONVIERTE EL ESTADO EN UN INDICE UNICO
        s1, s2, s3 = state
        return s1 * 9 + s2 * 3 + s3

    def _get_real_params(self, state):
        # MAPEA EL ESTADO DISCRETO
        s1_weight_level, s2_rep_level, s3_fatigue_level = state

        # MAPEO DE PESO RELATIVO S1 A % DEL 1RM
        weight_percentages = [0.80, 0.85, 0.90, 0.95]
        target_weight = self.F_max *
weight_percentages[s1_weight_level]

        # MAPEO DE REPETICIONES S2
        target_reps = [5, 10, 15][s2_rep_level]

        # MAPEO DE FATIGA S3 A UN FACTOR DE PENALIZACION DE LA FUERZA
        fatigue_factor = 1.0 - (s3_fatigue_level * 0.1)

        return target_weight, target_reps, fatigue_factor

    def step(self, action):

        # EL ENTORNO EJECUTA UNA ACCION Y DEVUELVE EL NUEVO ESTADO, LA
RECOMPENSA Y EL EPISODIO TERMINO

        planned_weight, planned_reps, fatigue_factor =
self._get_real_params(self.current_state)
```

```
        # CÁLCULO DE LA RECOMPENSA Y EL FALLO
        prob_failure = planned_weight / (self.F_max * fatigue_factor)
        stimulus = planned_weight * planned_reps

        reward = 0
        is_failure = prob_failure > 1.0 or random.random() <
(prob_failure - 1.0)

        if is_failure:

            # PENALIDAD POR FALLO/SOBREENTRENAMIENTO
            reward = -5
            self.F_max -= (self.F_max * 0.001)
            next_fatigue = min(self.current_state[2] + 1, 2)

        else:
            # ENTRENAMIENTO EXITOSO
            max_possible_stimulus = self.F_max * 0.95 * 15

            if stimulus > (max_possible_stimulus * 0.70):
                # RECOMPENSA MÁXIMA POR PROGRESO
                reward = 10
                delta_fmax = self.k_gain * stimulus
                self.F_max += delta_fmax
                next_fatigue = min(self.current_state[2] + 1, 2)

            else:
                # RECOMPENSA MODERADA
                reward = 3
                next_fatigue = max(self.current_state[2] - 1, 0)

        # ACTUALIZACIÓN DE ESTADO (Transición)
        s1_weight_level, s2_rep_level, _ = self.current_state

        # MANTENER
```

```
        if action == 0:
            next_state = (s1_weight_level, s2_rep_level, next_fatigue)

        # SUBIR REPETICIONES
        elif action == 1:
            next_state = (s1_weight_level, min(s2_rep_level + 1, 2),
next_fatigue)

        # SUBIR CARGA
        elif action == 2:
            next_state = (min(s1_weight_level + 1, 3), s2_rep_level,
next_fatigue)

        # BAJAR CARGA
        elif action == 3:
            next_state = (max(s1_weight_level - 1, 0), s2_rep_level,
next_fatigue)

        # SUBIR AMBOS REPETICION Y CARGA
        elif action == 4:
            next_state = (min(s1_weight_level + 1, 3),
min(s2_rep_level + 1, 2), next_fatigue)

        else:
            raise ValueError("Acción no válida")

        self.current_state = next_state
        done = True

        return self._state_to_index(next_state), reward, done, {}
```

PASO II EL AGENTE DE Q_LEARNING
IMPLEMENTA EL ALGORITMO Q-LEARNING PARA ENTRENAR UN AGENTE

```
    def q_learning_agent(env, num_episodes=10000):


        # HIPERPARAMETROS
        alpha = 0.1
        gamma = 0.9
        epsilon = 1.0
        epsilon_decay_rate = 0.0001
        min_epsilon = 0.01

        Q_table = defaultdict(lambda: np.zeros(env.action_space_size))
        fmax_history = []

        print(f"Comenzando el entrenamiento por {num_episodes} episodios
(días de entrenamiento)...")

        for episode in range(num_episodes):
            state = env.reset()
            done = False

            while not done:
                # ELEGIR ACCION: EPSILON-GREEDY
                if random.random() < epsilon:
                    action = random.randrange(env.action_space_size)
                else:
                    action = np.argmax(Q_table[state])

                next_state, reward, done, _ = env.step(action)

                # ACTUALIZAR LA Q-TABLE ECUACION DE BELLMAN
                old_value = Q_table[state][action]
                next_max = np.max(Q_table[next_state])
                new_value = (1 - alpha) * old_value + alpha * (reward +
gamma * next_max)
                Q_table[state][action] = new_value

                state = next_state

            epsilon = max(min_epsilon, epsilon - epsilon_decay_rate)
            fmax_history.append(env.F_max)

            if (episode + 1) % 1000 == 0:
                print(f"Episodio: {episode + 1}/{num_episodes}. F_max
actual: {env.F_max:.2f} kg.")

        print("Entrenamiento finalizado.")
        return Q_table, fmax_history, env.F_max
```

PASO III EJECUCION, ANALISIS Y VISUALIZACION DETALLADA
ANALIZA LA Q-TABLE Y MUESTRA LA MEJOR ACCION PARA CADA ESTADO, EJECUTA UNA SIMULACION CON LA POLITICA FINAL APRENDIDA.

```
# INICIALIZAR

hip_thrust = HipThrustTrenInferior(initial_fmax=100.0)

# ENTRENAR
Q_table_final, fmax_historia, fmax_final =
q_learning_agent(hip_thrust, num_episodes=10000)

print("-" * 50)
print(f"| FUERZA INICIAL (1RM): {hip_thrust.initial_fmax:.2f} kg")
print(f"| FUERZA FINAL (1RM):   {fmax_final:.2f} kg")
print(f"| GANANCIA TOTAL:      {fmax_final -
hip_thrust.initial_fmax:.2f} kg")
print("-" * 50)

# MAPEOS PARA UNA MEJOR LECTURA DE SALIDAS
rep_map = {0: '5', 1: '10', 2: '15'}
fatigue_map = {0: 'Baja', 1: 'Media', 2: 'Alta'}
action_map = {0: 'Mantener', 1: 'Subir Reps', 2: 'Subir Carga', 3:
'Bajar Carga', 4: 'Subir Ambos'}
weight_map = {0: '80%', 1: '85%', 2: '90%', 3: '95%'}

# FUNCIÓN PARA INTERPRETAR LA POLÍTICA ÓPTIMA

def print_optimal_policy(Q_table, env):

    print("Política Óptima del Agente (Mejor Acción a Tomar):")

    policy_output = []

    for s1 in range(4):
        for s2 in range(3):
            for s3 in range(3):
                state_tuple = (s1, s2, s3)
                state_index = env._state_to_index(state_tuple)

                if state_index in Q_table:
                    best_action_index =
np.argmax(Q_table[state_index])

                    policy_output.append(
                        f"| Peso: {weight_map[s1]} / Reps:
{rep_map[s2]} / Fatiga: {fatigue_map[s3]} "
                        f"-> Mejor Acción:
**{action_map[best_action_index]}**"
                    )

    for line in sorted(policy_output):
        print(line)

print_optimal_policy(Q_table_final, hip_thrust)

# FUNCIÓN DE BITÁCORA DE SESIONES

def run_simulation(env, Q_table, days=20):

    env.reset()
```

```
    print("\n" + "="*80)
    print("|ENTRENAMIENTO DE TREN INFERIOR HIP THRUST (Días con
Política Óptima) |")
    print("="*80)

    header = "| Día | Fatiga | 1RM Inicial | Acción Elegida | Peso
Planificado | Repeticion Planificadas | Recompensa (R) | 1RM Final |"
    print(header)
    print("-" * 80)

    for day in range(1, days + 1):

        current_state_tuple = env.current_state
        state_index = env._state_to_index(current_state_tuple)

        # Elegir la mejor acción (Explotación pura)
        action_index = np.argmax(Q_table[state_index]) if state_index
in Q_table else 0

        fmax_before = env.F_max
        planned_w, planned_r, _ =
env._get_real_params(current_state_tuple)

        # Ejecutamos el step para obtener el nuevo estado y la
recompensa
        _, reward, _, _ = env.step(action_index)

        # Imprimir la bitácora
        print(
            f"| {day:<3} | "
            f"{fatigue_map[current_state_tuple[2]]:<6} | "
            f"{fmax_before:<11.2f} | "
            f"{action_map[action_index]:<14} | "
            f"{planned_w:<16.2f} | "
            f"{planned_r:<17} | "
            f"{reward:<14} | "
            f"{env.F_max:<9.2f} |"
        )

    print("-" * 80)

# LLAMADA A LA FUNCIÓN DE BITÁCORA
run_simulation(hip_thrust, Q_table_final, days=30)

# GRÁFICO DE PROGRESO DE LA FUERZA

plt.figure(figsize=(10, 6))
plt.plot(fmax_historia, label='Fuerza Máxima (F_max) en kg')
plt.title('Evolución de la Fuerza Máxima (1RM) con el Entrenamiento
RL')
plt.xlabel('Episodios (Días de Entrenamiento)')
plt.ylabel('Fuerza Máxima (kg)')
plt.grid(True, linestyle='--')
plt.axhline(hip_thrust.initial_fmax, color='r', linestyle=':',
label='Fuerza Inicial')
plt.legend()
plt.show()
```