Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingenieria Civil Eléctrica

Proyecto final

Estabilización de la operación de molino SAG mediante modelo prescriptivo utilizando técnicas de Deep Reinforcement Learning Offline

Nombre: José Luis Cádiz Sejas
Profesor de cátedra: Javier Ruiz del Solar
Profesor auxiliar: Francisco Leiva
Fecha de realización: 29 de agosto de 2023
Fecha de entrega: 29 de agosto de 2023
Santiago de Chile

Índice de Contenidos

Índice de Figuras

1. Introducción y contexto

La motivación de este proyecto es contribuir a una operación más eficiente del proceso productivo del cobre mediante técnicas de **Reinforcement Learning** (RL) [1]. Específicamente dentro de las plantas concentradoras de cobre, centrándose en un activo critico que define en gran medida la productividad de la planta, el **molino SAG** (molino semi-autógeno). En la figura 1 se puede observar dentro de que etapa del proceso productivo del concentrado de cobre se encuentra el molino SAG.

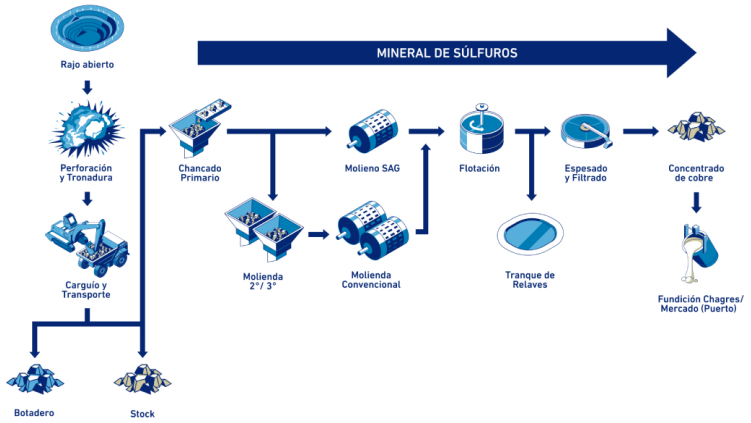

Figura 1: Proceso productivo del concentrado de cobre.

Para mejorar el entendimiento del proceso operativo del molino SAG, se debe entender el objetivo de este, el cual consiste en seguir disminuyendo el tamaño de las partículas del mineral proveniente del área de chancado. Para esto el mineral se mezcla con agua y cal, para luego reducir su tamaño gracias a la acción de las mismas partículas que poseen diversos tamaños y por el movimiento de numerosas bolas de acero que se desplazan en caída libre cuando el molino gira. En la figura 2 se puede observar una imagen real de un molino SAG.

Figura 2: Molino SAG de 11 metros de diámetro.

Debido a que el material entrante al molino SAG posee diversas propiedades, se deben adoptar estrategias operativas acorde al contexto operacional. Variables clave que definen que estrategia adoptar lo son la **granulometría**, **dureza** y **porcentaje de arcillas** del mineral. Según esto se

adoptan estrategias que consisten en definir **rpm**, **porcentaje de sólidos** y **celda de carga** (nivel de carga) del molino.

Debido a los altos costos de inversión, la experimentación en tiempo real para mejorar la toma de decisiones no es posible. Por lo tanto, el enfoque típico de Reinforcement Learning que interactúa con el entorno no es factible. Sin embargo, existe un nuevo enfoque de RL llamado **Reinforcement Learning Offline** [2], que se caracteriza por aprender políticas óptimas únicamente a partir de datos históricos. Esta política aprendida posteriormente puede ser re-entrenada (fine-tuning) en un ambiente Online para seguir mejorando su aprendizaje pero con un conocimiento previo del proceso [3]. En la figura 3 se esquematiza esta idea.

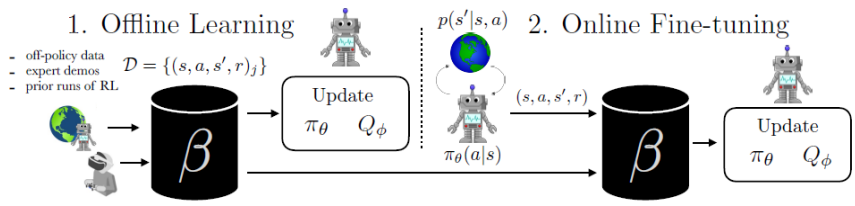

Figura 3: Esquema de entrenamiento Offline y posterior fine-tuning con interacción Online.

El problema a resolver busca encontrar una política óptima que defina el límite de celda de carga que debe tener el molino SAG a través del tiempo con el objetivo de que las toneladas por horas producidas (**TPH**) se mantenga estable.

Las caídas de TPH identificadas se deben principalmente por dos motivos:

i) **Caídas por acción del sistema de control**: Esto ocurre en contextos donde el mineral es muy grande o muy duro, en donde en general la decisión que se toma es disminuir el porcentaje de sólidos y aumentar las rpm, pero esto en algunas ocasiones genera que la celda de carga del molino supere o este muy cercano a su limite superior (**setpoint HH celda de carga**), lo cual activa el sistema de control, reduciendo la cantidad de mineral que ingresa al molino, lo que a su vez genera caídas de TPH respecto de su objetivo (**setpoint TPH**). Lo mencionado anteriormente se esquematiza en la figura 4.

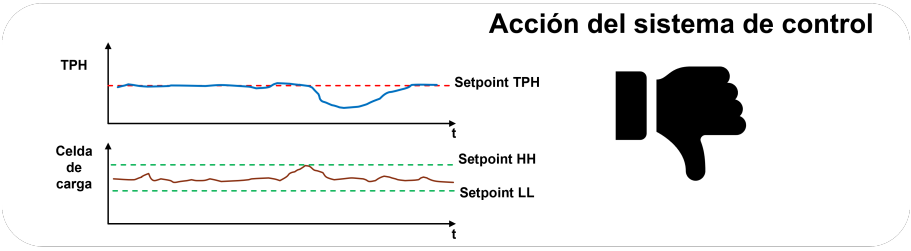

Figura 4: Esquema de caídas de TPH por acción del sistema de control.

En este contexto, la pregunta es si se podría haber subido el setpoint HH celda de carga del

molino con el objetivo de lograr el procesamiento del mineral pero sin este tipo de caídas, las cuales se caracterizan por ser de alto impacto respecto de las caídas por embancamiento.

ii) **Caídas por embancamiento**: Esto ocurre en contextos en que el molino se encuentra demasiado lleno, es decir con un alto nivel de celda de carga. es importante entender que este tipo de caídas de TPH no se deben a la acción del sistema de control, si no que es netamente por restricciones físicas del proceso. Lo mencionado anteriormente se esquematiza en la figura 5.

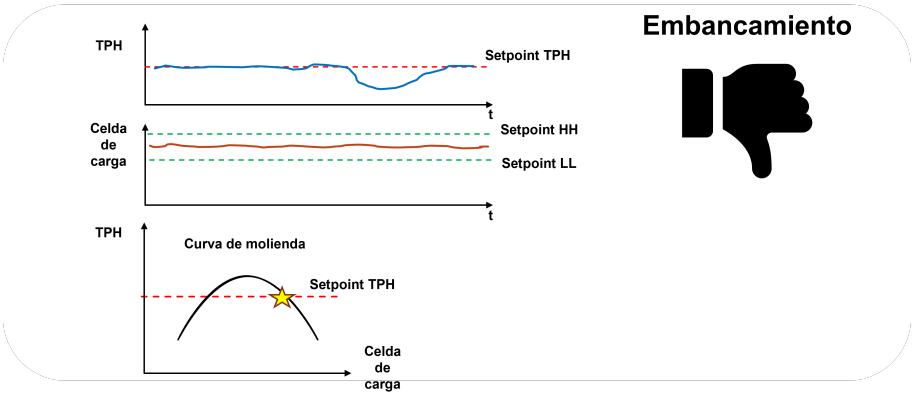

Figura 5: Esquema de caídas de TPH por embancamiento.

Para diferenciar a que se deben las caídas de TPH, se debe observar la celda de carga minutos previos a la caída, con el objetivo de visualizar si la celda de carga estaba demasiado cerca de su setpoint HH, si esto es así la caída se debe a la acción del sistema de control, si no es así, se debe a una caída por embancamiento del molino. Adicionalmente en la figura 6, se observa el caso óptimo en que no existen caídas de TPH.

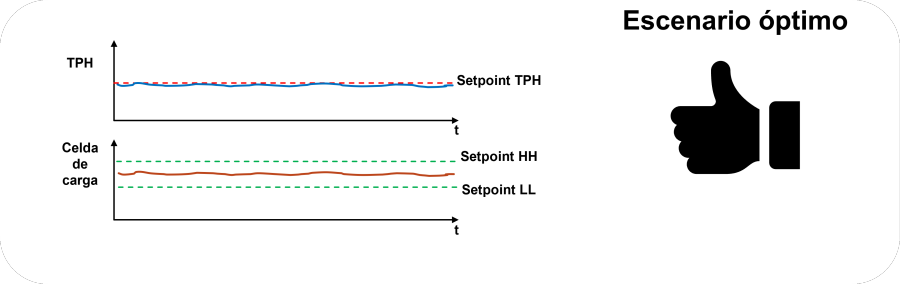

Figura 6: Escenario operativo óptimo.

Este proyecto se centrará solamente en las caídas de TPH por acción del sistema de control, pero para el proyecto de tesis se contemplaran ambos tipos de caídas.

2. Marco teórico

- **Reinforcement Learning** [1]: Es aprendizaje en base a prueba y error, se basa en el marco de los **Procesos de Decisión de Márkov** (MDP), Reinforcement Learning aborda el problema de cómo un **agente** activo y autónomo aprende **políticas óptimas** mientras interactúa con un **entorno o ambiente** inicialmente desconocido. El objetivo de resolver un MDP es encontrar la política óptima.

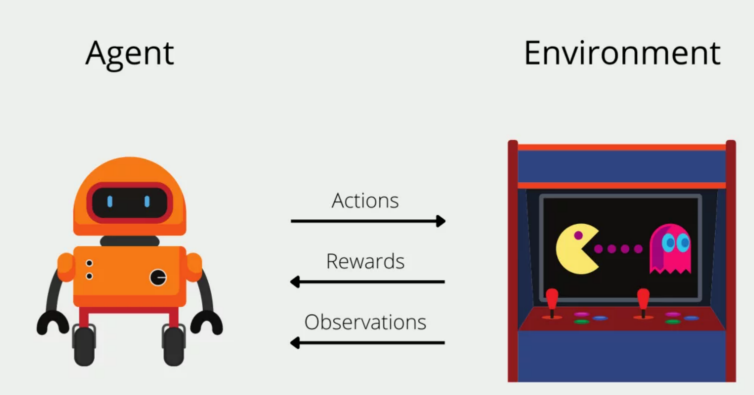

Figura 7: Ilustración Reinforcement Learning.

La interacción agente-ambiente da lugar a una distribución de probabilidad sobre secuencias de pares estado-acción, a las que se llaman **"trayectorias"**.

$$\underbrace{p_\pi(s_1, a_1, \ldots, s_T, a_T)}_{p_\pi(\tau)} = p(s_1) \prod_{t=1}^{T} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

El objetivo del Reinforcement Learning consiste en encontrar una política que maximice el valor esperado del **retorno (suma descontada de recompensas)** que recibe el agente con respecto a las trayectorias que visita en su interacción con el ambiente:

$$J_{RL}(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t) \right]$$

$$R = \sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t)$$

- **Conceptos básicos de Reinforcement Learning** [1]:

1. **Proceso de decisión de Márkov (MDP)**: En un MDP, la información necesaria para caracterizar interacciones agente-ambiente pasadas está contenida en el estado actual. Dichos estados se dice que poseen la **"propiedad de Márkov"**. **"Dado el presente, el futuro no depende del pasado"**. Un MDP esta definido por la tupla $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, donde:
   a) $\mathcal{S}$: Conjunto de estados.
   b) $\mathcal{A}$: Conjunto de acciones.
   c) $\mathcal{T}$: Función de transición de estados.
   d) $\mathcal{R}$: Función de recompensa.

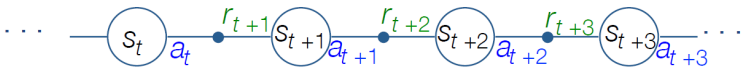

Figura 8: Ilustración de un Proceso de Decisión de Márkov (MDP).

   Formalización clásica para modelar problemas de toma de decisiones secuenciales. Para $t = 1, ..., T$:
   a) El agente se encuentra en un estado $s_t \in \mathcal{S}$.
   b) El agente elige y ejecuta una acción $a_t \in \mathcal{A}$ de acuerdo a su política $\Pi(a_t|s_t)$.
   c) El ambiente evoluciona a nuevo estado $s_{t+1} \in \mathcal{S}$ de acuerdo a la función de transición $\mathcal{T}(s, a, s') = p(s_{t+1}|s_t, a_t)$ y el agente recibe una recompensa escalar r_t, de acuerdo a la función de recompensa $\mathcal{R}(s, a)$.

2. **Agente**: Es una entidad (algoritmo) que interactúa con un entorno con el objetivo de aprender cómo tomar decisiones que maximicen alguna medida de recompensa a lo largo del tiempo. El agente ejecuta acciones de acuerdo a su política. La ejecución de estas acciones generan recompensas o castigos para el agente, y tienen consecuencias (a priori inciertas), dado que podrían afectar el estado del ambiente, y así, futuras acciones y recompensas.
3. **Hipótesis de recompensa**: Todo objetivo puede ser descrito a través de la maximización de la recompensa acumulada.
4. **Dilema de exploración y explotación**: Se refiere a la decisión que un agente debe tomar entre explorar nuevas opciones y acciones desconocidas o explotar las acciones que hasta ahora han demostrado ser efectivas en términos de recompensas.
5. **Recompensa**: Una recompensa r_t es una señal de retroalimentación escalar que indica qué tan bien está despeñándose el agente en el paso t. La tarea del agente es maximizar su retorno (suma descontada de recompensas).
6. **Factor de descuento**: $\gamma \in [0, 1]$, es un parámetro utilizado para determinar cómo se ponderan las recompensas futuras en relación con las recompensas inmediatas. En otras palabras, el factor de descuento determina cuánto valor se le da a las recompensas a largo plazo en comparación con las recompensas que se pueden obtener en el siguiente paso.
   a) Si $\gamma = 0$, el agente solo se preocupa por la recompensa inmediata en el siguiente paso y no considera las recompensas futuras. Esto se llama un enfoque de "descuento cero".

b) Si $\gamma = 1$, el agente valora las recompensas futuras de la misma manera que las recompensas inmediatas. Esto implica una consideración completa de las recompensas a largo plazo.

7. **Estado**: El estado del ambiente s_t es la representación interna que tiene el agente del ambiente, es decir, es la representación del entorno observable para el agente y utiliza esta información para seleccionar la próxima acción a tomar. Los estados pueden ser discretos o continuos.
8. **Acción**: las acciones a_t son las decisiones que toma un agente en un entorno para interactuar y lograr sus objetivos. Las acciones representan las elecciones que el agente puede hacer en cada paso de tiempo para influir en el estado del entorno y, en última instancia, obtener recompensas. Las acciones pueden ser discretas o continuas.
9. **Política**: La política del agente es una estrategia o un conjunto de reglas que dicta cómo el agente debe seleccionar acciones en función de las observaciones del entorno. Puede ser determinista o estocástica, dependiendo de si la política siempre toma la misma acción en una situación dada o si tiene cierta incertidumbre. Matemáticamente, una política Π es una distribución de acciones dado los estados:

$$\Pi(a|s) = P[A_t = a|S_t = s]$$

Las políticas de los MDP dependen del estado actual, no de la historia, es decir, las políticas son estacionarias (independientes del tiempo), es decir, $A_t \sim \Pi(\cdot|S_t)$, $\forall t > 0$.

10. **Función de valor** $V^\pi(s)$: La función de valor V asigna un valor numérico a cada estado en el espacio de estados del entorno, indicando cuán deseable o beneficioso es estar en ese estado. Esta función refleja la cantidad de recompensa acumulada que un agente puede esperar recibir en el futuro a partir de un estado dado, siguiendo una determinada política.

$$V^\pi(s) = \mathbb{E}_{\tau_t \sim p_\pi(\tau_t)} \left[\sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k) \middle| s_t = s \right]$$

11. **Función** $Q^\pi(s, a)$: La función Q es utilizada para medir el valor esperado acumulado que un agente puede obtener al tomar una determinada acción en un estado particular y luego seguir una política específica para el resto de la secuencia de acciones. El uso de funciones de valor estado-acción elimina la necesidad de almacenar políticas de manera explícita o de conocer el modelo (probabilidades de transición de estado).

$$Q^\pi(s, a) = \mathbb{E}_{\tau_t \sim p_\pi(\tau_t)} \left[\sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k) \middle| s_t = s, a_t = a \right]$$

12. **Ecuaciones de Bellman**: Para resolver un problema de Reinforcement Learning, se debe considerar el retorno que el agente podría recibir de acuerdo a las acciones que este ejecuta. Notando que $J_{RL}(\pi) = \mathbb{E}_{s_1 \sim p(s)}[V^\pi(s_1)]$, buscamos una política que tenga asociada una función de valor óptima.

$$V^*(s) = \max_\pi V^\pi(s)$$

$$\vee$$

$$Q^*(s, a) = \max_\pi Q^\pi(s, a)$$

Existe una relación directa entre $V^\pi(s)$ y $Q^\pi(s,a)$:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(a|s)}[Q^\pi(s,a)]$$

$$Q^\pi(s,a) = r(s,a) + \gamma\mathbb{E}_{s'\sim p(s'|s,a)}[V^\pi(s')]$$

De lo anterior, se pueden establecer relaciones de recurrencia para $V^\pi(s)$ y $Q^\pi(s,a)$, las cuales se conocen como **"ecuaciones de Bellman"**:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(a|s)}\left[r(s,a) + \gamma\mathbb{E}_{s'\sim p(s'|s,a)}[V^\pi(s')]\right]$$

$$Q^\pi(s,a) = r(s,a) + \gamma\mathbb{E}_{s'\sim p(s'|s,a)}\left[\mathbb{E}_{a'\sim\pi(a'|s')}[Q^\pi(s',a')]\right]$$

Tanto $V^*(s)$ como $Q^*(s,a)$ satisfacen su respectiva ecuación de Bellman, no obstante, en este caso pueden ser escritas de manera especial considerando la definición de las funciones de valor óptimas. Estas ecuaciones se conocen como **"ecuaciones de optimalidad de Bellman"**:

$$V^*(s) = \max_{a\in A}\left[r(s,a) + \gamma\mathbb{E}_{s'\sim p(s'|s,a)}[V^*(s')]\right]$$

$$Q^*(s,a) = r(s,a) + \gamma \max_{a'\in A}\left[\mathbb{E}_{s'\sim p(s'|s,a)}[Q^*(s',a')]\right]$$

Métodos matemáticos clásicos para lograr aproximar estas funciones de manera eficiente lo son la **programación dinámica** y los **métodos de Monte Carlo**:

Inicializar π
Inicializar umbral $\epsilon > 0$ y $\Delta v \leftarrow 0$
Inicializar $V^\pi(s)$ (aleatoriamente), pero con $V^\pi(s_{\text{terminal}}) \leftarrow 0$
while $\Delta v < \epsilon$ **do**
    $\Delta v \leftarrow 0$
    foreach $s \in \mathcal{S}$ **do**
        $v_s \leftarrow V^\pi(s)$
        $V^\pi(s) = \max_{a\in\mathcal{A}}\left[r(s,a) + \gamma\mathbb{E}_{s'\sim p(s'|s,a)}V^\pi(s')\right]$
        $\Delta v \leftarrow \max\{\Delta v, |v_s - V^\pi(s)|\}$
    end
end

Figura 9: Ejemplo de algoritmo de programación dinámica.

Inicializar una política $\pi(s)$
Inicializar $Q^{\pi}(s, a)$ aleatoriamente
Inicializar $S(s, a) \leftarrow 0$ y $N(s, a) \leftarrow 0$ para cada $s \in \mathcal{S}, a \in \mathcal{A}$
for *episodio = 1, N* **do**
  Seleccionar aleatoriamente (s_1, a_1)
  Generar *rollout* siguiendo política π, desde (s_1, a_1): $s_1, a_1, r_1, s_2, ...s_{T-1}, a_{T-1}, r_{T-1}, s_T$
  $R \leftarrow 0$
  for *t=T-1, 1* **do**
    $R \leftarrow r_t + \gamma R$
    if $(s_t, a_t) \notin \{s_1, a_1, ..., s_{t-1}, a_{t-1}\}$ **then**
      $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$
      $S(s_t, a_t) \leftarrow S(s_t, a_t) + R$
    end
  end
  $Q^{\pi}(s, a) \leftarrow \frac{S(s,a)}{N(s,a)}$ para todo $s \in \mathcal{S}, a \in \mathcal{A}$
  $\pi(s_t) \leftarrow \arg\max_{a \in \mathcal{A}} Q^{\pi}(s_t, a)$
end

Figura 10: Ejemplo de algoritmo de Monte Carlo.

- **Taxonomía de algoritmos de Reinforcement Learning**: Los algoritmos de RL pueden ser clasificados dentro de 4 categorías, siendo no siempre excluyentes una de la otra.

  1. **Value-Based**: Aproximan $V^{\star}(s)$ o $Q^{\star}(s, a)$ para derivar una política.
  2. **Policy-Based**: Buscan $\Pi(a|s)$ a través de la optimización directa de $J_{RL}(\pi)$.
  3. **Actor-Critic**: Aproximan conjuntamente $V^{\star}(s)$ o $Q^{\star}(s, a)$ y una política $\Pi(a|s)$.
  4. **Model-Based & Model-Free**: Hacen o no uso de un modelo del ambiente.

  Por ejemplo, **Q-Learning** y **SARSA** son algoritmos **Value-based & Model-Free**, ya que aproximan funciones Q y no necesitan un modelo del entorno. **REINFORCE** es un ejemplo de algoritmo **Policy-Based & Model-Free**, ya que representa su política $\Pi_{\theta}(a|s)$ explícitamente a través una red neuronal optimizando sus parámetros θ mediante gradiente ascendente según $\nabla J_{RL}(\pi_{\theta})$. Por otro lado, el algoritmo **Actor-Critic**, es de la familia que indica su nombre, esto debido a que posee una red neuronal para aproximar funciones Q, llamado **"Critic"** y otra red neuronal que aprende políticas óptimas llamado **"Actor"**. Ver figura 11 y 12 para facilitar el entendimiento de la taxonomía de estos algoritmos.

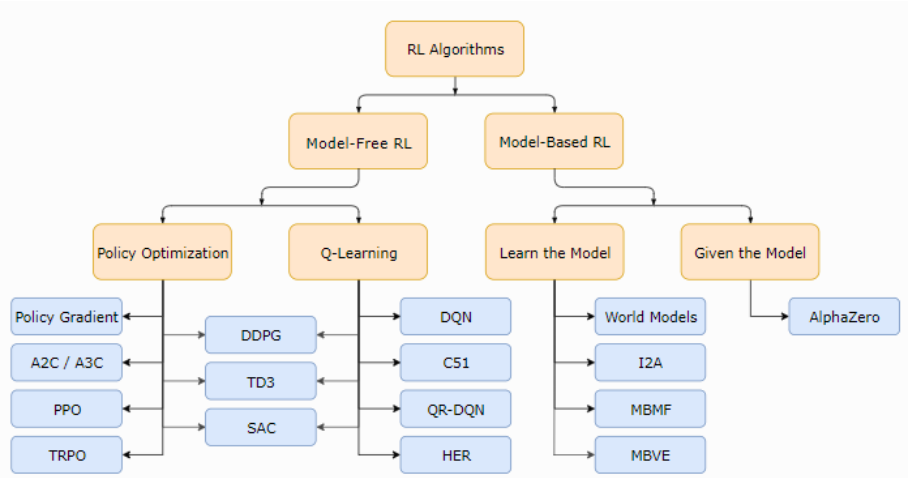

Figura 11: Taxonomía de algoritmos de Reinforcement Learning (OpenAI).

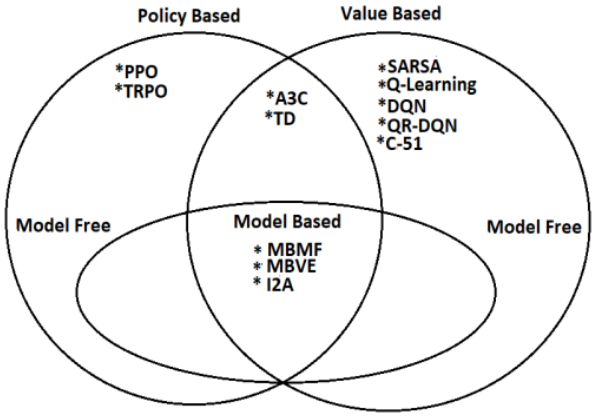

Figura 12: Taxonomía de algoritmos de Reinforcement Learning en diagrama de Venn.

- **Algoritmos de Reinforcement Learning**:

  1. **Value-Based**: Dentro de los algoritmos Value-Based tenemos los algoritmos por aprendizaje temporal conocidos como **Temporal-Difference Learning (TD-Learning)**, estos son una combinación de ideas entre programación dinámica y métodos de Monte Carlo. **Q-Learning** y **SARSA** son ejemplos de algoritmos **TD-Learning**, en la figura 13 e 14 se pueden ver sus algoritmos respectivos.

Inicializar $Q^{\pi}(s, a)$ aleatoriamente
Inicializar parámetro $\alpha \in (0, 1]$ y $\epsilon > 0$
for *episodio = 1, N* **do**
  Obtener s_1
  for *t=1, T* **do**
    Con probabilidad ϵ, elegir a_t aleatoriamente, si no, $a_t = \arg\max_{a \in \mathcal{A}} Q_\theta(s_t, a)$
    Ejecutar acción a_t, observar r_t y s_{t+1}
    $Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha(r_t + \gamma \text{ máx}_{a'} Q^{\pi}(s_{t+1}, a') - Q^{\pi}(s_t, a_t))$
  end
end

Figura 13: Algoritmo Q-Learning.

Inicializar $Q^{\pi}(s, a)$ aleatoriamente
Inicializar parámetro $\alpha \in (0, 1]$ y $\epsilon > 0$
for *episodio = 1, N* **do**
  Obtener s_1
  Con probabilidad ϵ, elegir a_1 aleatoriamente, si no, $a_1 = \arg\max_{a \in \mathcal{A}} Q_\theta(s_1, a)$
  for *t=1, T* **do**
    Ejecutar acción a_t, observar r_t y s_{t+1}
    Con probabilidad ϵ, elegir a_{t+1} aleatoriamente, si no, $a_{t+1} = \arg\max_{a \in \mathcal{A}} Q_\theta(s_{t+1}, a)$
    $Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha(r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) - Q^{\pi}(s_t, a_t))$
  end
end

Figura 14: Algoritmo SARSA.

Es importante mencionar el concepto de **"Deep Reinforcement Learning"**, este surge de la combinación entre el **Deep Learning** y el **Reinforcement Learning**. El deep RL se caracteriza por el uso de redes neuronales artificiales como aproximadores funcionales. El primer trabajo que popularizó el uso de redes neuronales en RL fue **Deep Q-Network (DQN)**. En la figura 15 se muestra una ilustración que plasma la diferencia de ideas entre Q-Learning y DQN. Por otro lado, en la figura 16 se puede apreciar el detalle de su algoritmo.

DQN emplea una red neuronal $Q_\theta(s, a)$ para aproximar $Q^*(s, a)$. Esta red neuronal, $Q_\theta(s, a)$, es entrenada empleando una variante de Q-Learning que posee dos elementos fundamentales, un **Experience Replay** y el uso de **Target Networks**.

Las experiencias (s_t, a_t, r_t, s_{t+1}) son almacenadas en un buffer finito $\mathcal{D}$. Batches de experiencias son muestreados uniformemente de $\mathcal{D}$ para actualizar los parámetros de $Q_\theta(s, a)$, según:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[(y - Q_{\theta_i}(s, a)) \nabla_{\theta_i} Q_{\theta_i}(s, a) \right], \text{ donde:}$$
$$y = r(s, a) + \gamma \max_{a' \in A} \mathbb{E}_{s' \sim p(s'|s,a)} Q_{\theta_{i-k}}(s', a')$$

Lo anterior permite la reutilización de experiencias, y también romper la correlación que existe entre ellas.

Se utiliza una copia $Q_{\bar{\theta}}(s, a)$ de $Q_\theta(s, a)$ para el computo de y. Esta copia es actualizada

copiando los parámetros de $Q_\theta(s, a)$ cada k > 1 actualizaciones. Este mecanismo reduce la posibilidad de oscilaciones (o divergencia) en los parámetros θ, al introducir un "delay" en el efecto que tienen los cambios en $Q_\theta(s, a)$ en el computo de y.

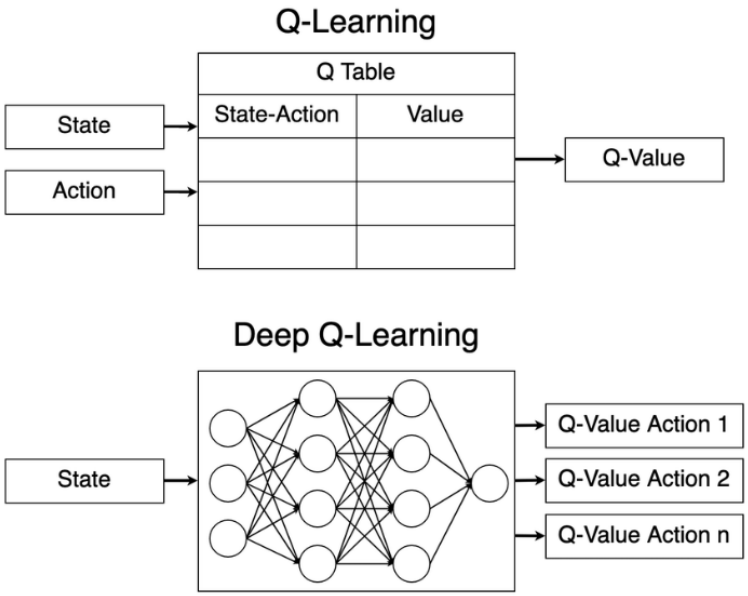

Figura 15: Diagrama Q-Learning vs Deep Q-Learning.

Inicializar $Q_\theta(s, a)$ con parámetros θ
Inicializar $Q_{\overline{\theta}}(s, a)$ con parámetros $\overline{\theta} \leftarrow \theta$
Inicializar *replay memory* D
for *episodio = 1, M* **do**
  Obtener s_1
  for *t=1, T* **do**
    Con probabilidad ϵ, elegir a_t aleatoriamente, si no, $a_t = \arg\max_{a \in \mathcal{A}} Q_\theta(s_t, a)$
    Ejecutar acción a_t, observar r_t y s_{t+1}
    guardar transición (s_t, a_t, r_t, s_{t+1}) en D
    Muestrear un *minibatch* de N transiciones (s_j, a_j, r_j, s_{j+1}) de D
    Calcular $y_j = \begin{cases} r_j & \text{si } s_{j+1} \text{ es un estado terminal,} \\ r_j + \gamma \max_{a'} Q_{\overline{\theta}}(s_{j+1}, a') & \text{si no.} \end{cases}$
    Actualizar $Q_\theta(s, a)$ minimizando el costo $L(\theta) = \frac{1}{N} \sum_{j=1}^{N} (y_j - Q_\theta(s_j, a_j))^2$
    Cada C actualizaciones de $Q_\theta(s, a)$, $\overline{\theta} \leftarrow \theta$
  end
end

Figura 16: Algoritmo DQN.

2. **Policy-Based**: Como se menciono anteriormente, los algoritmos de esta familia se caracterizan por aprender distribuciones de probabilidad de acciones dados los estados a través de una red neuronal, mejorando su política mediante gradiente ascendente. En la figura 17 se plasma la idea de que la salida de la red es una distribución de probabilidad, yendo más al detalle, la red entrega una media y desviación estándar que son parámetros de algún tipo

de distribución, como por ejemplo una Gaussiana. En la figura 18 se muestra el algoritmo **REINFORCE**.

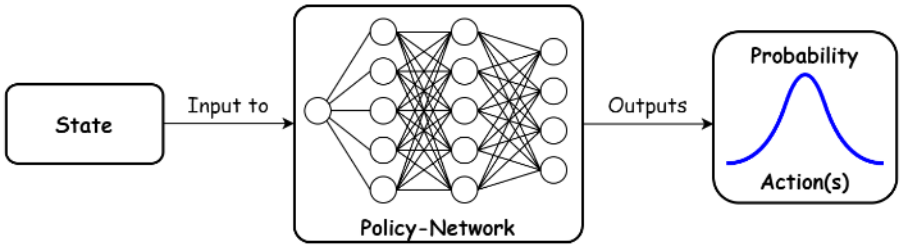

Figura 17: Diagrama de acciones generadas en algoritmos Policy-Based.

Inicializar $\pi_\theta(a|s)$ con parámetros θ
for *i=1, M* **do**
  for *k=1, N* **do**
    Obtener s_1
    for *t=1, T-1* **do**
      Ejecutar acción $a_t \sim \pi_\theta(a_t|s_t)$, observar r_t y s_{t+1}
      Guardar transición (s_t, a_t, r_t) en $\tau^{(k)}$
    end
  end
  Calcular $\nabla_\theta J_{\text{RL}} \approx \frac{1}{N}\sum_{k=1}^{N}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log\left(\pi_\theta\left(a_t^{(k)}|s_t^{(k)}\right)\right)\right)\left(\sum_{t=1}^{T}\gamma^{t-1} r\left(s_t^{(k)}, a_t^{(k)}\right)\right)\right]$
  Actualizar $\pi_\theta(a|s)$ con gradiente ascendente según $\nabla_\theta J_{\text{RL}}(\pi_\theta)$
end

Figura 18: Algoritmo REINFORCE.

3. **Actor-Critic**: Esta familia de algoritmos es una mejora de los algoritmos Policy-Based al reducir la alta varianza de la estimaciones del gradiente mediante la incorporación del termino **"Advantage Function"**:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

$$A^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s'\sim p(s'|s,a)} V^\pi(s') - V^\pi(s)$$

$$A^\pi(s,a) \approx r(s,a) + \gamma V^\pi(s') - V^\pi(s)$$

con esto:

$$\nabla_\theta J_{RL}(\pi_\theta) \approx \frac{1}{N}\sum_{k=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log\left(\pi_\theta\left(a_t^{(k)}|s_t^{(k)}\right)\right)\underbrace{\left(Q^\pi\left(s_t^{(k)}, a_t^{(k)}\right) - V^\pi\left(s_t^{(k)}\right)\right)}_{A^\pi(s^{(k)}, a^{(k)})}\right)$$

Advantage Function es una medida que cuantifica cuán favorable es tomar una acción en comparación con las acciones promedio disponibles en un estado específico bajo una política dada. Según su signo tiene la siguiente interpretación:

a) **Acciones más favorables**: Si $A^\pi(s,a) > 0$, significa que la acción a es mejor de lo que se esperaría de las acciones promedio en ese estado bajo la política Π. En este caso, tomar la acción a es más ventajoso que tomar las acciones promedio.

b) **Acciones menos favorables**: $A^{\pi}(s,a) < 0$, la acción a es peor de lo que se esperaría de las acciones promedio en ese estado bajo la política Π. En este caso, tomar la acción a es menos ventajoso que las acciones promedio.

Bajo este concepto surge la necesidad de aproximar $V^{\pi}(s)$ mediante $V_{\phi}(s)$, esto a través de una red neuronal denominada **"Critic"**, por otro lado, la red neuronal encargada de aprender la distribución de probabilidad de acciones dado los estados, es denominada **"Actor"**. En la figura 19 se muestra un ejemplo sencillo del algoritmo Actor-Critic.

Inicializar $\pi_\theta(a|s)$ con parámetros θ
Inicializar $V_\phi(s)$ con parámetros ϕ
for *i=1, M* **do**
  for *k=1, N* **do**
    Obtener s_1
    for *t=1, T-1* **do**
      Ejecutar acción $a_t \sim \pi_\theta(a_t|s_t)$, observar r_t y s_{t+1}
      Guardar transición (s_t, a_t, r_t) en $\tau^{(k)}$
    end
  end
  Ajustar ϕ minimizando $L(\phi) = \frac{1}{TN}\sum_{k=1}^{N}\sum_{t=1}^{T}\left(V_\phi\left(s_t^{(k)}\right) - \sum_{t'=t}^{T}\gamma^{t'-t} r\left(s_{t'}^{(k)}, a_{t'}^{(k)}\right)\right)^2$
  Calcular $\nabla_\theta J_{\mathrm{RL}} \approx \frac{1}{N}\sum_{k=1}^{N}\left[\sum_{t=1}^{T}\nabla_\theta \log\left(\pi_\theta\left(a_t^{(k)}|s_t^{(k)}\right)\right) A^{\pi}\left(s_t^{(k)}, a_t^{(k)}\right)\right]$
  Actualizar $\pi_\theta(a|s)$ con gradiente ascendente según $\nabla_\theta J_{\mathrm{RL}}(\pi_\theta)$
end

Figura 19: Algoritmo Actor-Critic.

- **Clasificación de algoritmos de Reinforcement Learning según temporalidad de aprendizaje**: Los algoritmos que se preocupan por la política que generó decisiones pasadas de estado-acción se conocen como algoritmos **On-Policy** [1], mientras que aquellos que la ignoran son conocidos como **Off-Policy** [1].

  Los algoritmos de aprendizaje Off-Policy evalúan y mejoran una política que es diferente de la política utilizada para la selección de acciones.

  Los métodos On-Policy intentan evaluar o mejorar la política que se utiliza para tomar decisiones. En contraste, los métodos Off-Policy evalúan o mejoran una política diferente de la utilizada para generar los datos.

  Dicho de otra forma, los algoritmos Off-Policy realizan actualizaciones siguiendo una política diferente a la que se esta siguiendo. Por otro lado, los algoritmos On-Policy realizan actualizaciones usando la misma politica que estan siguiendo.

  Por ejemplo, Q-Learning es Off-Policy pues actualiza Q siguiendo una politica greedy pero la politica que genera las transiciones es epsilon-greedy. En contraste, SARSA es On-Policy pues es consistente con la estrategia epsilon-greedy al momento de realizar actualizaciones.

  En la figura 20 (a) se observa que las experiencias generadas a través de interacciones agente-ambiente son utilizadas para actualizar la política. tras actualizar Π_k se deben generar nuevos datos para poder realizar una nueva actualización, Las experiencias para actualizar a Π_k son generadas por Π_k.

En la figura 20 (b) las interacciones agente-ambiente son guardadas en un replay buffer $\mathcal{D}$. Experiencias de $\mathcal{D}$ son muestreadas para realizar actualizaciones de la política. De este modo $\mathcal{D}$ presenta experiencias generadas por Π_0, ...Π_k.

Por otro lado, los algoritmos **Offline** (**Offline Reinforcement Learning** [2]) hacen referencia a aprender políticas sin interacciones agente-ambiente, las experiencias provienen de un dataset fijo $\mathcal{D}$. En principio cualquier algoritmo Off-Policy puede ser transformado a un algoritmo Offline, por ejemplo, aplicando Q-Learning a un dataset fijo (buffer).

Desafíos del Offline Reinforcement Learning, son la exploración, el dataset $\mathcal{D}$ es fijo, por lo que la exploración queda fuera de lo que un algoritmo de Offline RL puede abordar. Además, obtener un dataset $\mathcal{D}$ que sea representativo puede ser muy complejo. En la figura 20 (c) las experiencias para aprender una política son muestreadas de un buffer $\mathcal{D}$. Las experiencias en $\mathcal{D}$ fueron generadas por una política Π_β (que puede ser desconocida). No existe interacción agente-ambiente durante el aprendizaje de Π.

En Offline RL se requiere formular y responder preguntas del tipo "¿Qué hubiese ocurrido si...?" pues se quiere obtener una política Π que sea mejor que Π_β, es decir, queremos aprender algo mejor que lo que se observa en $\mathcal{D}$. Lo anterior se traduce en un problema de cambio de distribución entre la política que recopiló los datos y la política aprendida (**Distributional shift**).

Distributional shift se refiere a la discrepancia entre las distribuciones de datos que se utilizan para entrenar el modelo y las distribuciones de datos que se encuentran en el ambiente en el que el modelo se desplegará. En muchos casos, los datos históricos pueden provenir de un entorno o una política pasada que difiere significativamente del entorno actual. Esta discrepancia entre las distribuciones de datos puede resultar en una mala generalización y un rendimiento deficiente cuando se despliega el modelo en el mundo real.

El problema de distributional shift puede llevar a que el modelo tenga dificultades para adaptarse a las nuevas situaciones y tomar decisiones precisas en el entorno actual. Esto se debe a que el modelo no ha visto suficientes ejemplos de las situaciones actuales durante su entrenamiento, lo que puede resultar en comportamientos inesperados o subóptimos.

Para abordar este problema, es importante desarrollar métodos de Offline Reinforcement Learning que sean más resistentes al cambio en la distribución y que puedan generalizar de manera efectiva a situaciones no vistas previamente. Esto puede implicar el uso de técnicas como **importance sampling** para tener en cuenta las diferencias en las distribuciones de datos y el uso de algoritmos que sean menos sensibles a los cambios en la distribución de datos.

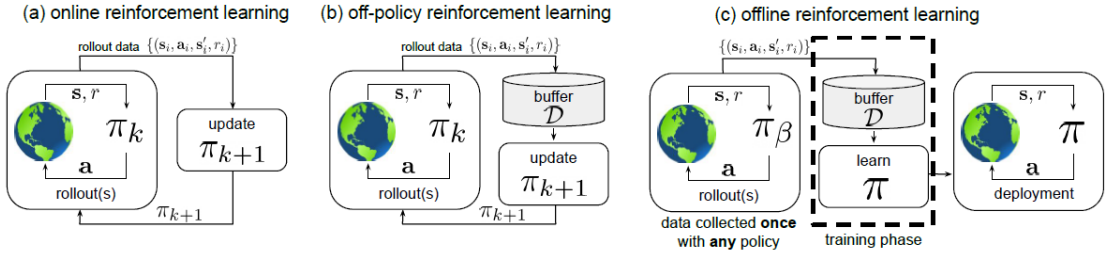

Figura 20: Ilustración de Reinforcement Learning On-Policy (a), Reinforcement Learning Off-Policy (b) y Reinforcement Learning Offline (c) [2].

- **Formulación del problema asociado a Reinforcement Learning Offline**: A continuación se explican los algoritmos **Conservative Q-learning (CQL)** y **Advantage-Weighted-Actor-Critic (AWAC)**.

  1. **Conservative Q-Learning (CQL)** [4]: CQL es la adaptación de DQN en un ambiente Offline, esto debido a que utilizar directamente algoritmos de RL Value-Based Off-Policy existentes en un entorno Offline generalmente resulta en un rendimiento deficiente, producto de que se generan problemas relacionados con el inicio de acciones fuera de la distribución y el overfitting. Esto suele manifestarse como estimaciones de la función de valor erróneamente optimistas. Si, en cambio, se pudiese aprender una estimación conservadora de la función de valor, que proporcione un límite inferior sobre los valores reales, este problema de sobre-estimación podría resolverse.

     Dado un batch de transiciones $\beta = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$ definiendo la regla de actualización de la función Q como:

     $$L(\beta, \theta) = \sum_{i=1}^{N} \left(r_i + \gamma \max_{a' \in A} Q_\theta(s'_i, a') - Q_\theta(s_i, a_i) \right)^2$$

     En CQL se busca modificar este objetivo agregando una penalización $\mathcal{C}(B, \theta)$ para obtener un nuevo objetivo de minimización, dado por:

     $$\tilde{L}(\beta, \theta) = L(\beta, \theta) + \alpha \mathcal{C}(\beta, \theta)$$

     Ejemplo de penalización:

     $$\mathcal{C}(\beta, \theta) = \mathbb{E}_{s \sim \beta} \left[\mathbb{E}_{a \sim \mu(a|s)} \left[Q_\theta(s, a) \right] \right]$$

     Con una correcta elección de $\mu(a|s)$, seria posible reducir Q-values altos, y mantener aproximaciones razonables para aquellos asociados a acciones que están dentro de la distribución.

     La elección $\mu(a|s) \propto \exp(Q_\theta(s, a))$ permite que la estimación regularizada de la función Q sea una cota inferior de la función Q real.

     En la practica la elección anterior de $\mathcal{C}(\beta, \theta)$ puede resultar en sub-estimaciones muy grandes, por lo que otra opción es definirla de manera diferente:

     $$\mathcal{C}(\beta, \theta) = \mathbb{E}_{s \sim \beta} \left[\mathbb{E}_{a \sim \mu(a|s)} \left[Q_\theta(s, a) \right] \right] - \mathbb{E}_{(s,a) \sim \beta} \left[Q_\theta(s, a) \right]$$

Esto permite que valores Q altos puedan ser asignados a pares (s, a) que estén dentro de la distribución.

2. **Advantage Weighted Actor Critic (AWAC)** [3]: AWAC es de la familia Actor-Critic basado en TD3. Este algoritmo nace a partir de la necesidad de tener procesos de entrenamiento más eficientes en ambientes complejos.

   Para esto el agente se entrena preliminarmente con un dataset $\mathcal{D}$ Offline, de modo que el modelo pueda posteriormente ser desplegado y ser sometido a fine-tuning pero esta vez interactuando con el ambiente de una manera Online.

   La política óptima se aprende a partir de la optimización de la siguiente función objetivo:

$$J(\phi) = \mathbb{E}_{s_t, a_t \sim \mathcal{D}} \left[\log \pi_\phi(a_t|s_t) \exp\left(\frac{1}{\lambda} A^\pi(s_t, a_t)\right) \right]$$

   En la figura 21 se puede ver en detalle su algoritmo.

Dataset $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)_j\}$
Initialize buffer $\beta = \mathcal{D}$
Initialize π_θ, Q_ϕ
for iteration $i = 1, 2, \ldots$ **do**
  Sample batch $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r) \sim \beta$
  $y = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}', \mathbf{a}'}[Q_{\phi_{k-1}}(\mathbf{s}', \mathbf{a}')]$
  $\phi \leftarrow \arg\min_\phi \ \mathbb{E}_{\mathcal{D}}[(Q_\phi(\mathbf{s}, \mathbf{a}) - y)^2]$
  $\theta \leftarrow \arg\max_\theta \ \mathbb{E}_{\mathbf{s}, \mathbf{a} \sim \beta} \left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda} A^{\pi_k}(\mathbf{s}, \mathbf{a})\right) \right]$
  if $i >$ num_offline_steps **then**
    $\tau_1, \ldots, \tau_K \sim p_{\pi_\theta}(\tau)$
    $\beta \leftarrow \beta \cup \{\tau_1, \ldots, \tau_K\}$
  end if
end for

Figura 21: Algoritmo AWAC [3].

3. Metodología

El algoritmo utilizado para aprender políticas óptimas es de la familia Actor-Critic, **AWAC**. Se hará uso de la librería de Offline RL **d3rlpy** [5] en lenguaje de programación Python. El modelo indicará el Setpoint HH celda de carga que se deberá configurar en el sistema de control 5 minutos después dado el estado actual del molino SAG. En la figura 22 se muestra un esquema del MDP que se desarrollara en el tiempo. A continuación se detalla la metodología a seguir:

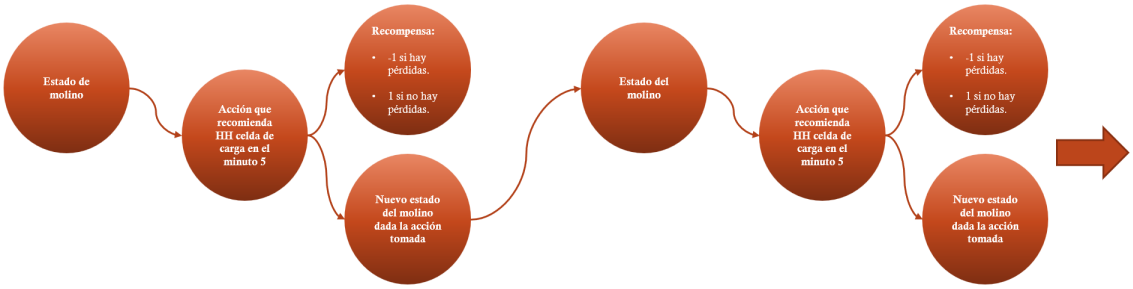

Figura 22: Diagrama MDP.

1. Se definen estados, acciones y reward.

   - **Estados**: Agua, porcentaje de sólidos, rpm, HH TPH, granulometría, celda de carga y Setpoint HH celda de carga.
   - **Acciones**: Setpoint HH celda de carga en minuto 5.
   - **Reward**: 1 si las pérdidas de TPH son menores a 100 y -1 si las pérdidas de TPH son mayores a 100.

2. El dataset es procesado para remover escenarios anómalos y se computan valores faltantes. Posteriomente, el dataset se divide en los conjuntos de entrenamiento y test, luego se estructuran los datasets en el formato que espera el modelo, para esto adicionalmente se deben generar labels que indiquen cuando termina cada episodio. En este caso de manera arbitraria los episodios serán los turnos en los cuales funciona la operación de la planta concentradora, es decir desde 12:00 hasta las 00:00.

3. Se entrena el modelo de RL con el dataset en formato MDP y se generan la predicciones de la recomendación de HH celda de carga en el conjunto de test.

4. Evaluación de las políticas aprendidas, se comparan las distribuciones de la política óptima generada por el agente y la acciones históricas tomadas a partir del dataset, esto para verificar que estén dentro de rangos factibles. Finalmente, se analizan algunas series de tiempo con el objetivo de verificar una mejora en la toma de decisiones del agente respecto de las decisiones que se tomaron en el escenario real.

4. Resultados

En la figura 23 se observa como converge el Loss del "Actor" luego de 10 épocas de entrenamiento y en la figura 24 se observa que el Loss del "Critic" no logra converger. Por otro lado, en la figura 25 se comparan las distribuciones de las acciones en data train vs la recomendación del agente, en donde se observa que las recomendaciones del agente están dentro del rango de la data de entrenamiento, sin embargo, existe una tendencia hacia niveles de carga más altos.

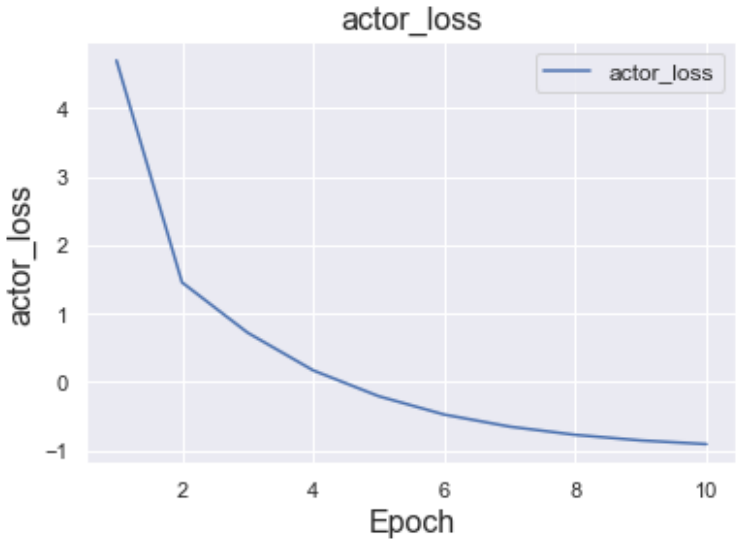

Figura 23: Loss Actor.

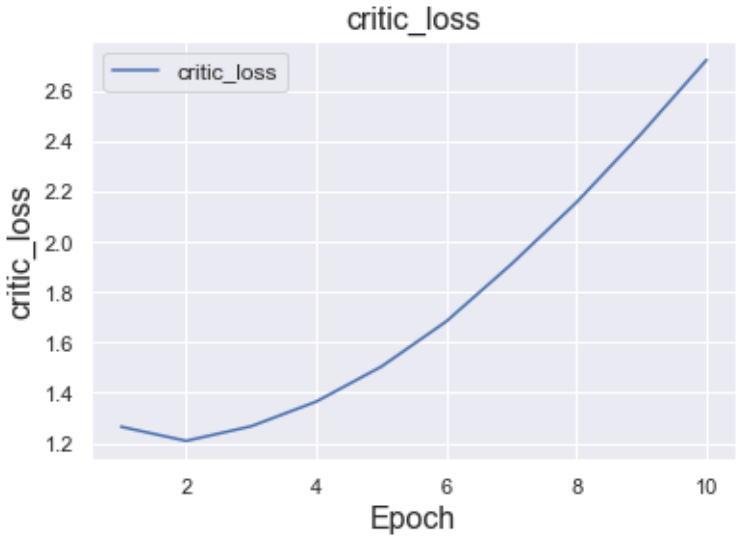

Figura 24: Loss Critic.

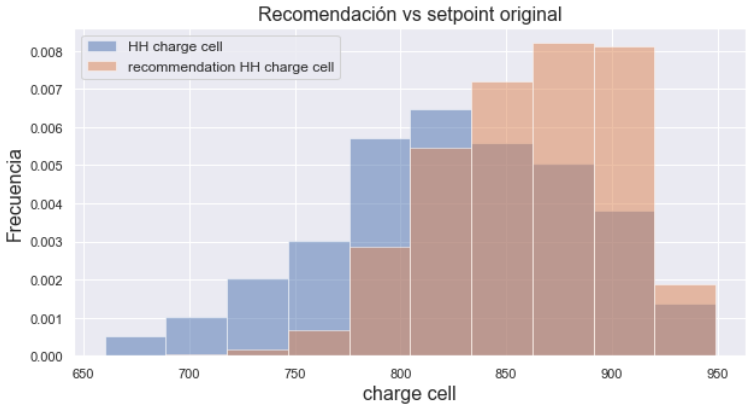

Figura 25: Comparación distribución data train vs recomendación.

Respecto a la data test, se comparan las distribuciones de las acciones tomadas vs la recomendación del agente en las fechas 2021/09 y 2020/04. En la figura 26 se observa que ambas distribuciones son similares, por lo que las recomendaciones del agente no son disruptivas, a partir de esto, en primera instancia no se sospecha alguna inconsistencia en ellas, sin embargo, en la figura 27 se observa que hay una gran diferencia entre ambas distribuciones, por lo que es importante entender a que se podría deber esto, en el análisis posterior se mencionará más sobre esto.

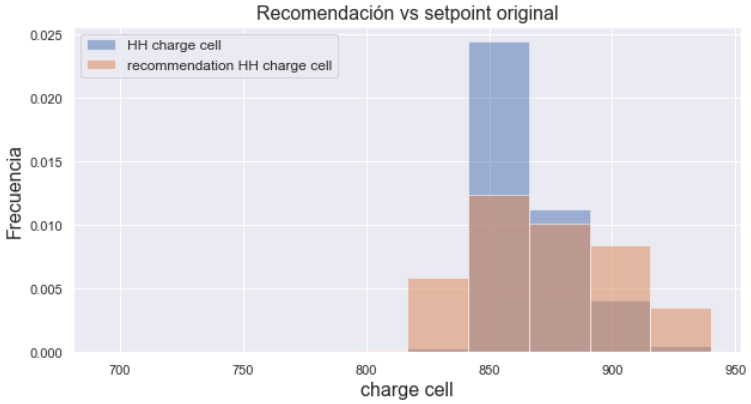

Figura 26: Comparación distribución data test 2021/09 vs recomendación.

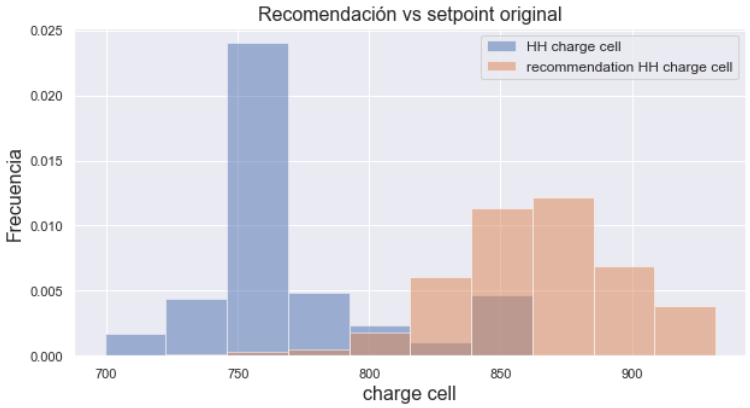

Figura 27: Comparación distribución data test 2020/04 vs recomendación.

En la figura 28, se observa como el modelo es capaz de anticipar los eventos de caída de TPH, recomendando que el setpoint de celda de carga sea aumentado para que el sistema de control no actúe.

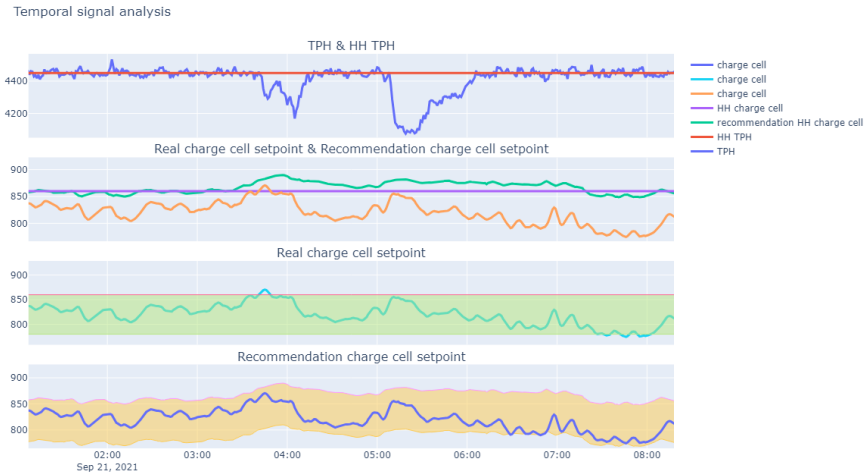

Figura 28: **Escenario inestable con buena recomendación**, TPH cae debido a que su celda de carga supera su setpoint y en consecuencia sistema de control actúa. La recomendación indica que se le de más espacio al molino para cargar, de este modo sistema de control no hubiese actuado.

En la figura 29 se observa como la recomendación del modelo es muy similar al setpoint real en el caso en que el TPH se encuentra estable.

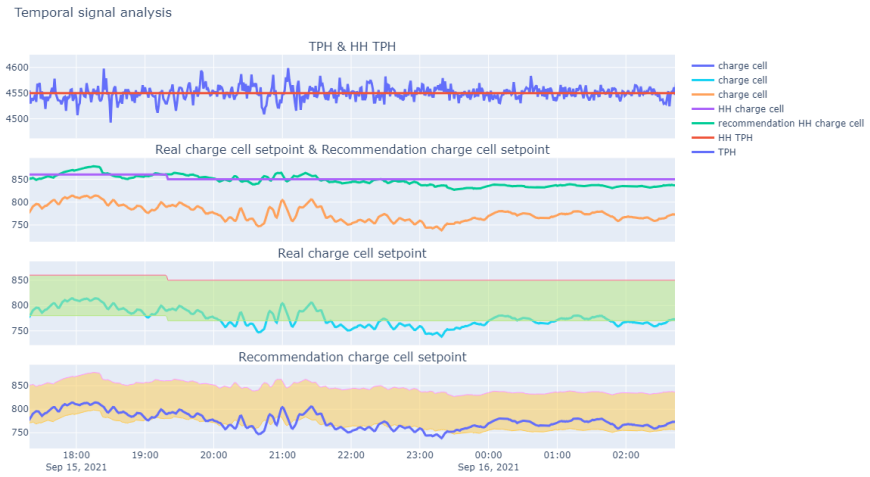

Figura 29: **Escenario estable con buena recomendación**, recomendación esta en torno a la decisión real que tomo la operación.

Por otro lado, en la figura 30, se evidencia un caso en que la recomendación del modelo no es correcta, ya que pese a que el sistema de control no actúa (su celda de carga no supera su setpoint) pero el molino SAG se embanca.

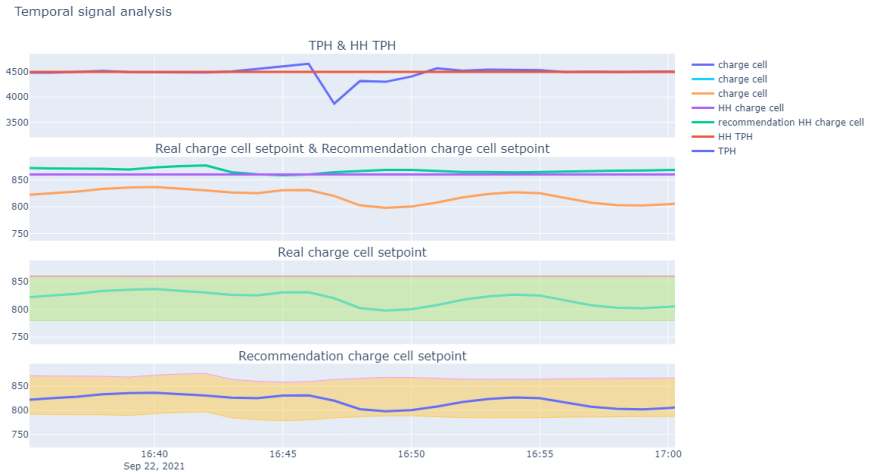

Figura 30: **Escenario inestable con mala recomendación**, recomendación indica seguir cargando el molino pero TPH cae pese a estar con una carga inferior. No actúa sistema de control, esta caída es asociada a embancamiento.

Finalmente, en la figura 31, se observa que la recomendación esta muy por encima de lo que se opero en ese momento, esto esta alineado con la comparación de distribuciones de la figura 27. Si bien lo que indica la recomendación tiene cierto sentido debido a que indica subir el setpoint HH, esto muy probablemente sea inviable en la practica. Profundizando en el estado del molino en ese periodo resulta que sus revestimientos se encontraban muy desgastados, lo que significa que el peso

del molino es mucho menor de lo usual, producto de esto, se desprende de que el agente no es capaz de captar el desgaste de los revestimiento de modo que indique una recomendación factible.

En definitiva, se necesita una mejor caracterización del nivel de desgaste de los revestimientos del molino, ya que se pueden generar recomendaciones fuera del alcance físico de este.

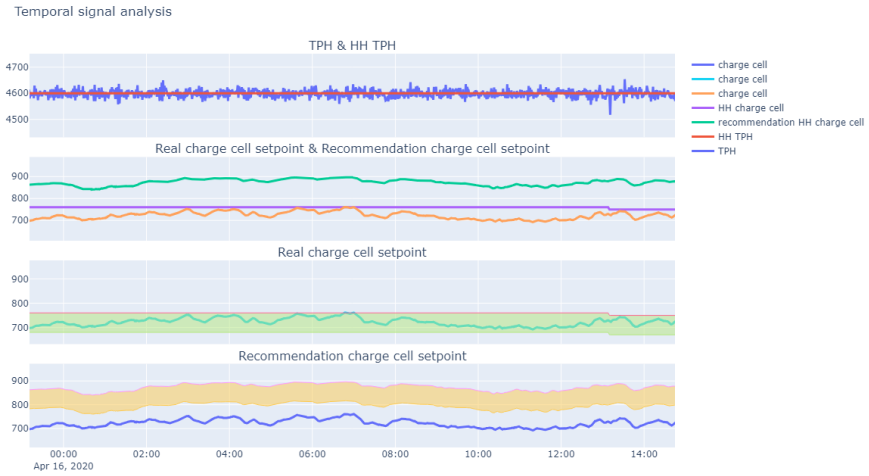

Figura 31: **Escenario de revestimientos desgastados**.

4.1. Conclusiones

El enfoque Offline del Reinforcement Learning permite aprender políticas óptimas a partir de data histórica sin necesidad de experimentar con la planta, sin embargo, surge la necesidad de poder testear las políticas aprendidas y para esto es necesario tener al menos un modelo de la planta que permita simular la incorporación de las políticas aprendidas dentro del sistema.

A partir de la necesidad de tener un modelo para poder testear las políticas aprendidas, se genera una cierta contradicción, ya que, si se tuviese un modelo de la planta, lo podríamos utilizar como ambiente y aprender políticas óptimas a partir de métodos convencionales de Reinforcement Learning.

Reinforcement Learning Offline es un método muy interesante para aprender políticas óptimas solamente a partir de data histórica, sin embargo, es necesario diseñar metodologías para poder testear de manera robusta las políticas aprendidas a partir del mismo tipo de estructura de información con la cual se entrena el algoritmo.

También es importante mencionar que para obtener resultados satisfactorios utilizando este enfoque de aprendizaje, es muy importante entender la dinámica del sistema que se busca optimizar, ya que es de vital importancia modelar de manera correcta dichas dinámicas como un proceso de decisión de Márkov.

Se concluye que los resultados han sido parcialmente satisfactorios ya que, si bien las políticas aprendidas son consistentes con los resultados esperados a partir del conocimiento previo, no se puede decir de manera categórica que las políticas aprendidas son óptimas o seguras debido a que no es posible la evaluación empírica.

Referencias

[1] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. The MIT Press, 2nd ed., 2018.

[2] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," arXiv preprint arXiv:2005.01643, 2020.

[3] A. Nair, A. Gupta, M. Dalal, and S. Levine, "Awac: Accelerating online reinforcement learning with offline datasets," arXiv preprint arXiv:2006.09359, 2020.

[4] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," Advances in Neural Information Processing Systems, vol. 33, pp. 1179–1191, 2020.

[5] T. Seno and M. Imai, "d3rlpy: An offline deep reinforcement learning library," The Journal of Machine Learning Research, vol. 23, no. 1, pp. 14205–14224, 2022.