Facultad de Ciencias

Departamento de Matemáticas

Enfoques de Bajo Rango para Resolver MDPs y Problemas de RL

Autor:

Diego Andrés Gómez Polo

Asesor:

Mauricio Junca

2022

Periodo Académico 2022-20

Abstract

This work studies some consequences of the low-rank hypothesis in the context of MDPs and Reinforcement Learning in general, from recent works such as those of [Zhu et al., 2021], [Agarwal et al., 2020] and [Uehara et al., 2021] to implementations of the algorithms described in those papers with some proposed modifications. The theoretical complexities of these algorithms, their assumptions and practical limitations, and the experimental results obtained in each implementation will also be discussed.

Resumen

En el presente trabajo se estudian algunas consecuencias de la hipótesis de bajo rango en el contexto de MDPs y Aprendizaje por Refuerzo en general, desde trabajos recientes como los de [Zhu et al., 2021], [Agarwal et al., 2020] y [Uehara et al., 2021] hasta implementaciones de los algoritmos descritos en dichos papers con algunas modificaciones propuestas. Se hablará también de las complejidades teóricas de dichos algoritmos, sus supuestos y limitaciones prácticas, y de los resultados experimentales obtenidos en cada implementación.
[1]

[1]**Key Words:** Reinforcement Learning, MDP, Low-Rank Hypothesis.

Índice

1 Introducción y Motivación

El campo de aprendizaje por refuerzo o Reinforcement Learning (RL) ha sido sumamente estudiado desde hace mucho tiempo y en contextos muy heterogéneos. Desde sus inicios en el estudio del comportamiento animal hasta sus bases matemáticas en la teoría de control y optimización, han sido muchos los avances en este campo a lo largo de las décadas. Pero en los últimos años, con un acceso ha crecientes capacidades de cómputo, hemos visto como las fronteras de lo que se creía posible alcanzar con RL han sido rotas una y otra vez. A la fecha de este documento, presenciamos como, por ejemplo, en [Fawzi et al., 2022] logran mejorar los métodos conocidos para una operación tan fundamental como la multiplicación de matrices por medio de RL y redes neuronales profundas. También el caso reciente de ChatGPT, un modelo de lenguaje sumamente avanzado con resultados en generación de texto, muchas veces indistinguibles de los generados por un humano bien versado, fue entrenado usando RLHF (Reinforcement Learning from Human Feedback) como en [Stiennon et al., 2020], una técnica de entrenamiento de redes neuronales profundas que busca incorporar conocimientos previos del ambiente por medio de demostraciones humanas, para luego construir un modelo de costos con base a la información dada por las respuestas de humanos y luego encontrar una política óptima sobre el ambiente (que sería las respuestas de la red neuronal en este caso) que maximice el modelo de costos previamente creado. Es decir, dentro de todas las posibles respuestas que puede dar un modelo, intenta encontrar una forma de elegir aquellas que tienen más sentido de acuerdo a los ejemplos dados por los humanos y esto lo resuelve mediante RL.

Los anteriores solo son unos pocos ejemplos muy recientes de grandes avances en distintos campos que hacen uso de técnicas de RL, por ende, hoy más que nunca es un buen momento para estudiar este tema y continuar ayudando a avanzar el campo.

En este sentido, no es raro intentar encontrar la intersección de RL y las dinámicas de bajo rango, ya que estas últimas son muy comunes en los datos obtenidos del mundo real. Esto se evidencia en [Udell and Townsend, 2019], donde da razones matemáticas detrás del porqué esperar que en matrices de alta dimensionalidad obtenidas sobre datos del mundo real se presente un rango bajo. Por ende, modelos que usen matrices de transición de procesos reales, con alta probabilidad podrían beneficiarse del supuesto de que dichas matrices son de bajo rango.

En el contexto de RL y más específicamente de Procesos de Decisión de Markov (MDPs), se ha trabajado con este supuesto hace relativamente poco y la estructura subyacente de bajo rango ha tomado distintos nombres como *factorizing MDP* en [Rendle et al., 2010] o *linear MDP* en [Amani et al., 2022] pero en realidad todos estos representan la misma estructura subyacente del MDP.

Con esto en mente, se decidió buscar literatura y métodos recientes para resolver problemas de RL en el contexto de bajo rango y ver qué tipo de técnicas usaban para explotar la estructura del MDP. Y fue así como se encontraron los trabajos de [Zhu et al., 2021], que busca aproximar matrices de transición por medio de optimización convexa, o [Agarwal et al., 2020] y [Uehara et al., 2021] que directamente tratan el problema de MDPs de bajo rango por medio de aproximadores de funciones y algoritmos de planeación en el ambiente, los cuales logran una exploración eficiente para descubrir la estructura del MDP e, incluso, en el caso de [Uehara et al., 2021], balancear la exploración con explotación para encontrar a la vez políticas óptimas para

una función de costo conocida.

2 Preliminares

En esta sección se dará un poco de contexto con respecto a algunos de los conceptos teóricos usados por los distintos algoritmos que se comparan. De igual forma se requieren conceptos previos en algunas áreas ya que no es completamente auto-contenida porque ello implicaría una extensión inadecuada para este tipo de documento.

2.1 Optimización Convexa

Las técnicas de optimización convexa son claves a la hora de aproximar modelos con alguna estructura convexa o una aproximación convexa razonable. En general, si se tiene un problema de optimización como el siguiente:

$$\begin{aligned} &\text{mín } f_0(x) \\ &\text{s.a. } f_i(x) \leq 0, i = 1, \ldots, m \\ &\quad a_i^T x = b_i, i = 1, \ldots, n \end{aligned}$$

Entonces, dicho problema se dice convexo si para toda i, f_i es una función convexa. Estos problemas son particularmente tratables ya que muchas subclases de problemas convexos cuentan con soluciones en tiempo polinomial, mientras que el caso general de optimización es un problema *NP-Hard* y si adicionalmente se tienen funciones suaves, en algunos casos se pueden usar métodos eficientes basados en gradientes.

2.1.1 Lagrangiano Aumentado

Sea $g : \mathbb{R}^n \to \mathbb{R}$ una función convexa, $A \in \mathbb{R}^{m \times n}$ y $b \in \mathbb{R}^m$. Considere el problema de optimización con restricciones

$$\begin{aligned} &\text{mín } g(x) \\ &s.a.\ Ax = b \end{aligned}$$

Definimos el Lagrangiano asociado a este problema como

$$\mathcal{L}(x; y) = g(x) + \langle y, Ax - b \rangle$$

y para $\sigma > 0$ definimos el Lagrangiano aumentado

$$\mathcal{L}_\sigma(x; y) = g(x) + \langle y, Ax - b \rangle + \frac{\sigma}{2} \parallel Ax - b \parallel^2,$$

Nota: Los métodos para resolver problemas de optimización a través del Lagrangiano aumentado fueron introducidos por primera vez por Hestenes [Hestenes, 1969] y Powel [Powell, 1969].
Alternativamente se puede presentar el Lagrangiano aumentado como

$$\mathcal{L}_\sigma(x; y) = g(x) + \frac{\sigma}{2} \parallel Ax - b + y/\sigma \parallel - \frac{1}{2\sigma} \parallel y \parallel^2,$$

con un par de cálculos sencillos se puede ver que ambas presentaciones son equivalentes. Note que si tomamos $y = 0$ nos queda

$$\mathcal{L}_\sigma(x; 0) = g(x) + \frac{\sigma}{2} \parallel Ax - b \parallel^2,$$

y obtenemos el funcional de penalización clásico sujeto a la restricción $Ax = b$.
La ventaja de usar el Lagrangiano aumentado es que, gracias a la presencia del término $\langle y, Ax - b\rangle$, la solución exacta del problema de optimización se puede hallar sin necesidad de mandar σ al infinito, contrario a lo que ocurre con la penalización clásica la cual ocasiona un deterioro en el condicionamiento del sistema a resolver.

Otra de las razones para usar el Lagrangiano aumentado es que para este, al igual que el Lagrangiano clásico, sus puntos de silla coinciden con las soluciones del problema de optimización, más explícitamente: (Teorema 2.1 [Glowinski, 2008]) Un punto (x, y) es un punto de silla de $\mathcal{L}$ si y sólo si es un punto de silla de $\mathcal{L}_\sigma$ para todo $\sigma > 0$. Además x es una solución del problema de optimización y $Ax = b$.

Para saber más sobre el Lagrangiano aumentado, su implementación y algoritmos para hallar puntos de silla ver [Glowinski, 2008] y [Fortin and Glowinski, 1983].

2.1.2 Problema Dual y Condiciones KKT

Sea F una función tal que $F : \mathbb{R}^n \to \overline{\mathbb{R}}$. Definimos su conjugado dual F^* por

$$F^* : \mathbb{R}^n \to \overline{\mathbb{R}},$$

$$F^*(x) = \sup_{v \in \mathbb{R}^n} \{\langle v, x\rangle - F(v)\}.$$

Para una función $g : \mathbb{R}^n \to \overline{\mathbb{R}}$ dada, considere el siguiente problema de optimización general:

$$\min_{x \in \mathbb{R}^n} g(x).$$

para formular el problema dual consideramos una *función de perturbación* $\Phi : \mathbb{R}^n \times \mathbb{R}^m \to \overline{\mathbb{R}}$ la cual satisface $\Phi(x, 0) = g(x)$ para todo $x \in \mathbb{R}^n$. El problema conjugado dual se puede formular como:

$$\sup_{y \in \mathbb{R}^m} \Phi^*(0, y)$$

donde Φ^* es el conjugado de Φ. La siguiente desigualdad s

$$\sup_{y \in \mathbb{R}^m} -\Phi^*(0, y) \leq \inf_{x \in \mathbb{R}^n} g(x, 0)$$

siempre se satisface y la diferencia entre ambas soluciones se conoce como la *brecha dual*. En el caso de que el problema de optimización tenga restricciones no se considera directamente la función g, sino una función $\tilde{g} = g + I_r$ donde $I_r(x) = 0$ si x satisface las restricciones y ∞ de lo contrario. La formulación del problema dual es análoga.

Generalmente solucionar el problema dual únicamente provee una cota inferior para la solución, sin embargo si el problema satisface ciertas condiciones se puede garantizar que la brecha dual sea igual a 0.

Definición 2.1 (Subdiferencial)**.** Un vector $\hat{x}$ es un *subgradiente* de una función convexa g en un punto x si para todo z se cumple que $g(z) \geq g(x) + \langle \hat{x}, z - x \rangle$. Al conjunto de todos los subgradientes de g en un punto x se le conoce como el *subdiferencial de g en x* y es denotado por $\partial g(x)$. [Rockafellar, 1970]

Consideramos el problema de optimización con restricciones:

$$\min_{x \in \mathbb{R}^n} f(x)$$

sujeto a

$$h_i(x) \leq 0, i = 1, \ldots, m,$$
$$l_j(x) = 0, j = 1, \ldots, r.$$

Para que la solución del problema dual coincida con la solución del problema original, es suficiente que se cumplan las condiciones de Karush-Kuhn-Tucker (KKT):

- $0 \in \partial f(x) + \sum_{i=1}^{m} u_i \partial h_i(x) + \sum_{j=1}^{r} v_j \partial l_j(x)$.
- $u_i \cdot h_i(x) = 0$ para i.
- $h_i(x) \leq 0$, $l_j(x) = 0$ para todo i, j.
- $u_i \geq 0$ para todo i.

Estas condiciones se conocen como estacionariedad, holgura complementaria, factibilidad primal y factibilidad dual respectivamente.

Teorema 2.1. Si x^*, u^* y v^* satisfacen las condiciones KKT entonces x^* y u^*, v^* son soluciones del primal y el dual respectivamente. [Boyd and Vandenberghe, 2004]

En resumen, las condiciones KKT son un criterio suficiente para garantizar que la brecha dual es igual a 0.

2.2 Aprendizaje por Refuerzo y MDPs

El Aprendizaje por Refuerzo o RL por sus siglas en inglés es un marco de trabajo en donde se intenta modelar situaciones en las que un agente tiene que tomar decisiones para lograr un objetivo específico, codificado usualmente dentro de recompensas numéricas que el agente obtiene por cada acción tomada. Esto es lo suficientemente amplio como para que las técnicas basadas en Aprendizaje por Refuerzo sean aplicadas en una gran cantidad de campos como se expuso a detalle con anterioridad.

En este sentido, es conveniente formalizar esta idea de un agente que actúa en un ambiente y recibe recompensas, por ende, se introduce a continuación la definición de un MDP o Proceso de Decisión de Markov.

Definición 2.2 (Markov Decision Process (MDP))**.** Un MDP o Proceso de Decisión de Markov es un cuarteto $(\mathcal{X}, \mathcal{A}, \mathcal{R}, P)$ donde $\mathcal{X}$ es el espacio de estados y $\mathcal{A}$ es el espacio de acciones, ambos potencialmente infinitos. $\mathcal{R}$ es el espacio de recompensas el cual suele ser $\mathbb{R}$ y, por último, y probablemente lo más importante de esta definición, tenemos a la dinámica del sistema $P : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \times \mathcal{R} \to [0, 1]$ tal que para $s, s' \in \mathcal{X}$, $a \in \mathcal{A}$ y $r \in \mathcal{R}$, entonces $P(s, a, s', r)$ es la probabilidad de obtener la recompensa r si estando en un estado s el agente pasó a otro estado s' por medio de una acción a. Es decir, P es una distribución de probabilidad en el conjunto $\mathcal{X} \times \mathcal{A} \times \mathcal{X} \times \mathcal{R}$.

Esta definición por sí sola no dice mucho, pero si la ponemos dentro del contexto de Aprendizaje por Refuerzo empieza a cobrar sentido. En el Aprendizaje por Refuerzo tendremos dos actores principales. Uno que no podemos controlar, el cual es el ambiente, y el otro que esta a nuestra disposición, el agente. Estos dos actores interactúan en pasos discretos. En el tiempo 0 el ambiente expone un estado $s_0 \in \mathcal{X}$. Con base en este estado el agente toma una acción $a_0 \in \mathcal{A}$ por lo que el ambiente responde con un siguiente estado $s_1 \in \mathcal{X}$ y una recompensa $r_0 \in \mathcal{R}$. Aquí ya empezamos a ver cómo todo se va relacionando con un MDP, ya que podemos usar P como una interpretación de la distribución verdadera del ambiente que está generando recompensas y estados siguientes con base a las acciones que tome el agente. Si repetimos el proceso muchas veces podemos obtener una cadena $(s_0, a_0, r_0, s_1, a_1, ...s_n, a_n, r_n, s_{n+1})$ a la cual llamaremos una *trayectoria*.

En este punto el lector ávido probablemente ya notó que hay un problema con nuestra definición. Para que esta interpretación funcione, la distribución real sobre nuestras trayectorias no puede tener memoria, en el sentido de que solo le debe importar el estado inmediatamente anterior.

Más específicamente: $P(s_n, a_n, s_{n+1}, r_n|(s_0, a_0, r_0, s_1, a_1, ...s_n)) = P(s_n, a_n, s_{n+1}, r_n)$. Esta es la propiedad que hace al sistema *Markoviano*. Además, cabe resaltar que esto, formalmente, no es una imposición sobre el ambiente mismo, si no sobre la forma por la cual el ambiente expone sus estados al agente, es decir, sobre el conjunto $\mathcal{X}$. Dicho hecho se hace evidente al ver que si tenemos una trayectoria $(s_0, a_0, r_0, s_1, a_1, ...s_n, a_n, r_n, s_{n+1})$ podemos convertirla en una Markoviana de forma sencilla definiendo unos nuevos estados $\hat{s}_n = (s_0, s_1, ..., s_n)$ que vuelvan el sistema un MDP a fuerza bruta. Claramente esto no es lo ideal, porque al convertir un espacio de estados en Markoviano explota la cardinalidad de este mismo, ya que para cada posible paso h de nuestra cadena, estamos definiendo $|\mathcal{X}|^h$ estados, por lo que la cardinalidad final de todo el espacio nuevo para un proceso de máximo h pasos sería $\sum_{i=0}^{h} |\mathcal{X}|^i$. Por este importante hecho, en la práctica buscamos procesos que naturalmente exhiban Markovianidad para hablar de MDPs.

Con esta definición podemos empezar a hacer varias observaciones e introducir algunos conceptos importantes. Note que dado un estado inicial $s_0 \in \mathcal{X}$, si tenemos alguna forma de escoger una acción a, entonces podemos calcular la probabilidad de pasar de s_0 a s por medio de la acción a como $P_{\mathcal{X}}(s|\mathcal{X}_0 = s_0, \mathcal{A}_0 = a) = \int_{r \in \mathcal{R}} P(\mathcal{X}_1 = s, \mathcal{R}_0 = r|\mathcal{X}_0 = s_0, \mathcal{A}_0 = a)dr$. A estas probabilidades sobre las transiciones de estados le llamaremos *la dinámica de transición del sistema* y será el objeto de estudio más importante de este documento. Una observación que nos permitirá comparar los algoritmos a implementar es pensar en el caso en el que el espacio de estados es finito. Este no es un caso fuera de lo común y algunos de los benchmarks clásicos como Pacman o Gridworld presentan justamente esta situación. Si $\mathcal{X}$ es finito, para cada acción $a \in \mathcal{A}$ tendremos una dinámica de transición completamente determinada por la matriz de transición de una Cadena de Markov Discreta.

A continuación, presentaremos algunas definiciones sumamente importantes en el contexto de Reinforcement Learning.

Definición 2.3 (Política sobre un MDP)**.** Dado un MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{R}, P)$, una política π sobre $\mathcal{M}$ es una función que va del espacio de estados $\mathcal{X}$ al espacio de distribuciones sobre acciones $\Delta(\mathcal{A})$ donde si el agente sigue la política π, la probabilidad de que en un estado s el agente tome la acción a será igual a $\pi(s)(a)$.

Definición 2.4 (Función de Recompensa o Costo Total (Expected Returns))**.** Dado un MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{R}, P)$ y un episodio o trayectoria sobre $\mathcal{M}$ dada por $(s_0, a_0, r_0, s_1, a_1, ...s_n, a_n, r_n, s_{n+1})$, Un retorno esperado es una función sobre el vector $(r_0, r_1, ..., r_n)$ que determina cuánto se ganó en dicho episodio.

La más sencilla de estas funciones sería simplemente sumar los r_i, pero en la práctica esto rara vez se usa, debido a que los objetivos del Aprendizaje por Refuerzo no siempre concuerdan con una suma simple de las recompensas. Muchas veces queremos maximizar las recompensas a corto o largo plazo, dependiendo del caso. Para esto se introduce el concepto de recompensa descontada.

Definición 2.5 (Recompensa Descontada (Discounted Rewards))**.** Dadas recompensas $(r_0, r_1, ..., r_n)$ generadas en un episodio de un MDP $\mathcal{M}$, la recompensa descontada G está dada por $G = \sum_{i=0}^{n} \gamma^i \cdot r_i$ donde $\gamma \in \mathbb{R}$ es un parámetro del modelo. Además, las recompensas parciales G_i serán $G_i = \sum_{k=i}^{n} \gamma^{k-i} \cdot r_k$ tal que $G_i = r_i + \gamma \cdot G_{i+1}$

Esta definición se adapta tanto al caso donde nos interesa más el retorno inmediato como aquel donde es mejor los retornos futuros ya que si γ es cercana al cero, nos enfocamos más en recompensas inmediatas, mientras que si γ es un número mayor a 1, nos interesan más las recompensas futuras.

Con base a esto, podemos empezar a pensar en valores esperados de las recompensas. De esta forma surgen los siguientes conceptos que serán la base de algoritmos clásicos de Aprendizaje por Refuerzo.

Definición 2.6 (Value Function sobre una política)**.** Dado un MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{R}, P)$ y una política π sobre $\mathcal{M}$, el Value Function V_π de esta política, representa el valor esperado de la variable aleatoria G si el agente sigue la política π, es decir, el valor esperado de la Recompensa Descontada. Formalmente será: $V_\pi(s) = \mathbb{E}_\pi[G_i | \mathcal{X}_i = s]$, donde $\mathcal{X}_i$ es la variable aleatoria que representa el $i - esimo$ estado de un episodio sobre $\mathcal{M}$

Existen otro tipo de funciones que buscan calcular qué tan buena o mala es una política en en un estado dado. En esta misma linea existe otro concepto de Aprendizaje por Refuerzo llamado la Q-Function, la cual se relaciona estrechamente con la anterior Value Function.

Definición 2.7 (Q-Function sobre una política)**.** Dado un MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{R}, P)$ y una política π sobre $\mathcal{M}$, la Q-function Q_π es una función de $\mathcal{S} \times A$ en $\mathbb{R}$ que nos da el valor esperado de la recompensa descontada dado que estando en un estado s se tomó una acción a. Es decir, $Q_\pi(s, a) = \mathbb{E}_\pi[G_i | \mathcal{X}_i =, \mathcal{A}_i = a]$, donde $\mathcal{A}_i$ es la variable aleatoria que representa la $i - esima$ acción tomada por el agente en un episodio dado sobre $\mathcal{M}$.

Definición 2.8 (Política Optimal)**.** Una política $\hat{\pi}$ se dice mejor que otra política π si, para todo $x \in \mathcal{X}$ se tiene que $V_{\hat{\pi}}(x) \geq V_\pi(x)$. Una política se dice optimal si esta es mejor que todas cualquier otra política (note que el $\geq$ en la definición implica que puede existir más de una política optimal).

Existen dos aproximaciones clásicas basadas en programación dinámica para encontrar políticas optimales dentro de un MDP. Estas son *Value Iteration* y *Policy Iteration*. Policy Iteration intenta empezar con una política aleatoria e ir sucesivamente

mejorándolal hasta que el algoritmo converja a una política lo suficientemente buena. Por otro lado, Value Iteration busca el mayor valor posible que la Value Function puede obtener en cada estado, con base a esto, se puede sacar una política óptima. Ambas son aproximaciones buenas y no está claro en la literatura si una es mejor que la otra en términos generales. Los algoritmos 1 y 2 son los previamente descritos.

Algorithm 1 Value Iteration Algoritmo para MDPs

Require: $S, A, P, R, \gamma, \Theta$

Inicialice $V(s)\ \forall s \in S$, arbitrariamente
repeat
  for each state $s \in S$ **do**
    $V(s) \leftarrow max_{a \in A} \sum_{s' \in S} P(s'|s,a) \cdot [R(s,a,s') + \gamma \cdot V(s')]$
  end for
until $V(s)$ converja en Θ

return $V^*(s)$ para todo $s \in S$

Algorithm 2 Policy Iteration Algoritmo para MDPs

Require: S, A, P, R, γ

Inicializar política π y $V(s)$ para todo $s \in S$, arbitrariamente
repeat
  for each state $s \in S$ **do**
    $V(s) \leftarrow \sum_{s' \in S} P(s'|s,\pi(s)) \cdot [R(s,\pi(s),s') + \gamma \cdot V(s')]$
  end for
  for each state $s \in S$ **do**
    $\pi(s) \leftarrow argmax_{a \in A} \sum_{s' \in S} P(s'|s,a) \cdot [R(s,a,s') + \gamma \cdot V(s')]$
  end for
until π converja

return $\pi^*(s)$ para todo $s \in S$

2.3 Low-Rank MDPs

Dado un MDP $\mathcal{M} := (\mathcal{X}, \mathcal{A}, \mathcal{R}, P)$ con espacios de estados y acciones posiblemente infinitos, definimos lo siguiente:

Definición 2.9 (Operador de bajo rango en un MDP (definición 1 en [Agarwal et al., 2020])). Dado un operador $T : \mathcal{X} \times \mathcal{A} \to \Delta(\mathcal{X})$, se dice que este es de bajo rango o que admite una descomposición de bajo rango con rango d si existen 2 funciones $\phi : \mathcal{X} \times \mathcal{A} \to \mathbb{R}^d$ y $\mu : \mathcal{X} \to \mathbb{R}^d$ tales que:

$$\forall x, x' \in \mathcal{X}, a \in \mathcal{A}, T(x'|x,a) = \langle \phi(x,a), \mu(x') \rangle,$$

y, además, $\forall x, x' \in \mathcal{X}, a \in \mathcal{A}$ también se tiene que para toda función $g : \mathcal{X} \to [0,1]$ se cumple $\| \phi(x,a) \|_2 \leq 1$ y $\| \int_{\mathcal{X}} \mu(x) g(x) dx \|_2 \leq \sqrt{d}$

Abusando un poco de la notación llamaremos a la dinámica de transición inducida por P como $\mathbb{P}$ tal que $\mathbb{P} : \mathcal{X} \times \mathcal{A} \to \Delta(\mathcal{X})$ y $\mathbb{P}(x, a, x') = \int_{r \in \mathcal{R}} P(x, a, x', r)dr$ y nuestro MDP $\mathcal{M}$ será de bajo rango justamente si $\mathbb{P}$ cumple con la definición 2.9

Definición 2.10 (Low-Rank MDP)**.** Dado un MDP $\mathcal{M} := (\mathcal{X}, \mathcal{A}, \mathcal{R}, P)$, y $\mathbb{P}$ la dinámica de transición de $\mathcal{M}$ inducida por P, se dice que $\mathcal{M}$ es de bajo rango si $\mathbb{P}$ es un operador de bajo rango (Definición 2.9).

2.3.1 Tipos de MDPs de Bajo Rango

Mucha de la motivación detrás de definir un MDP de bajo rango viene del hecho de que este es un marco de trabajo muy expresivo y engloba muchos tipos de MDPs presentes en la literatura. En términos generales, se suele considerar que el hecho de cumplir la definición 2.10 implica que el modelo presenta un espacio de variables latentes $\mathcal{Z}$ para el cual μ y ϕ están modelando transiciones. Dicho espacio latente puede presentar diversos tipos de estructuras que permiten distintas interpretaciones de lo que está haciendo el modelo. A continuación, se verán algunas de las interpretaciones más comunes encontradas en la literatura con respecto al espacio $\mathcal{Z}$.

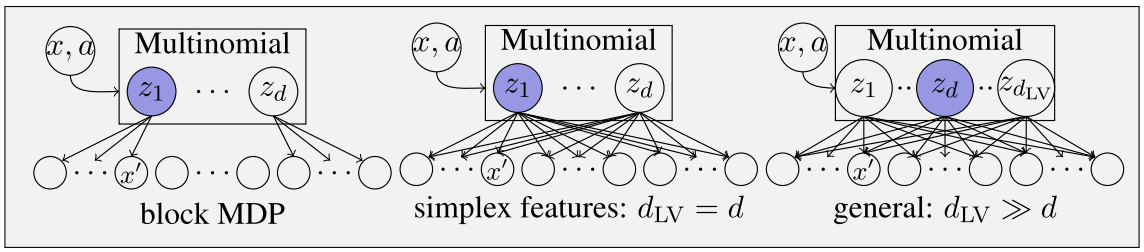


Figura 2.1: Algunos tipos de MDPs de bajo rango y su interpretación como transiciones en un espacio latente. Fuente original de esta imagen [Agarwal et al., 2020]

.

2.3.1.1 Simplex Features MDPs

Como notación, $d_{LV} := |\mathcal{Z}|$ es el número de variables latentes. En general el tamaño de los vectores de embedding puede ser mucho menor que el número de variables latente, como se muestra a la derecha de la figura 2.1 ($d_{LV} \gg d$). Este caso no es tan interpretable como otros ya que justamente corresponde al más general de todos. Sin embargo, si jugamos un poco con las condiciones de normalización expuestas en la definición 2.9, nos encontramos con situaciones más interesantes. Por ejemplo, note que podemos expresar μ como $(\mu_1, \ldots, \mu_d)$, tal que $\mu(x) := (\mu_1(x), \ldots, \mu_d(x)) \in \mathbb{R}^d$ y si $\mu_i \in \Delta(\mathcal{X})$ entonces $\| \int_{\mathcal{X}} \mu(x)dx \|_2 = \| (1, \ldots, 1) \|_2 = \sqrt{d}$. Es decir, si las entradas de μ se comportan como distribuciones sobre el espacio de estados $\mathcal{X}$, entonces se cumple una de las condiciones de normalización expuestas en la definición 2.9. Por otro lado, siempre se debe cumplir que $\int_{x' \in \mathcal{X}} \mathbb{P}(x, a, x')dx' = 1$, que es lo mismo que, $\int_{x' \in \mathcal{X}} \langle \phi(x, a), \mu(x') \rangle = 1$ y, por bilinealidad del producto punto, $\langle \phi(x, a), \int_{x' \in \mathcal{X}} \mu(x') \rangle = 1$, pero sabemos por hipótesis que esto es equivalente a $\langle \phi(x, a), 1_d \rangle = 1$, donde 1_d es el vector de unos d-dimensional, es decir, se debe cumplir que $\sum_{i=1}^{d} \phi_i(x, a) = 1$. Si, además, suponemos que para todo i, todo estado x y acción a, $\phi_i(x, a) \geq 0$, entonces

se da el caso descrito visualmente por la gráfica de la mitad de la figura 2.1, el caso de *Simplex Features*. Con el análisis previo queda bastante claro el porqué del nombre, ya que se tendrá que el rango de ϕ está contenido en el simplex d-dimensional, entonces, los valores de ϕ representan una distribución de probabilidad sobre d variables que, en la interpretación estándar, se toman justamente como el número de variables latentes del modelo, es decir, $d = d_{LV}$.

La explicación intuitiva de este caso es que se tiene un espacio de variables latentes por el cual el modelo siempre pasa antes de llegar al siguiente estado observable, entonces, habrán dos distribuciones intermedias que no podemos observar directamente: una primera distribución sobre $\mathcal{Z}$ para cada elemento de $\mathcal{X} \times \mathcal{A}$, que es la probabilidad de terminar en algún estado latente $z \in \mathcal{Z}$ dado que se parte de un estado x siguiendo una acción a; luego, la segunda distribución no conocida es la que describe el paso de $\mathcal{Z}$ a $\mathcal{X}$, por la cual, dado un estado objetivo x' se tiene una probabilidad de llegar a él partiendo de alguna variable latente z. Si le llamamos $p_i^{(x,a)}$ a la probabilidad de pasar a z_i partiendo desde x y tomando a y $q_i^{x'}$ a la probabilidad de llegar a x' al salir del estado latente z_i, entonces, $\mathbb{P}(x, a, x') = \sum_{i=1}^{d} p_i^{(x,a)} \cdot q_i^{x'} = \langle \phi(x, a), \mu(x') \rangle$

2.3.1.2 Block MDPs

Un MDP por bloque es un tipo de MDP con espacio de variables latentes para el cual las transiciones entre las variables latentes y el espacio de estados observable se presupone forman una partición de este mismo, es decir, para cada variable latente $z_i \in \mathcal{Z}$, existe un subconjunto de estados $\mathcal{X}_i := \{x \in \mathcal{X} : \mathbb{P}(x|z_i) > 0\}$ y forman una partición en el sentido que todo estado estará en exactamente uno de estos conjuntos (mire el diagrama de la izquierda de la figura 2.1). Este supuesto es muy estudiado en la literatura en trabajos como los de [Du et al., 2019] o [Misra et al., 2019]. Ahora bien, esta representación puede llegar a ser menos expresiva que el caso general de low-rank MDP [Agarwal et al., 2020], por ende, tanto este y como muchos otros trabajos se concentran principalmente en un low-rank MDP general.

3 Aprendizaje Con el Supuesto de Bajo Rango

Ya se ha visto en secciones anteriores argumentos que motivan considerar el supuesto de bajo rango en matrices que representan datos o dinámicas del mundo real. Además, se vio como para el caso específico de RL existe un marco teórico que ya incluye este supuesto y sobre el cual se puede trabajar (los MDPs de bajo rango). Por ende, a continuación se explorarán algunos resultados y algoritmos que pueden resultar útiles en este contexto.

Cabe destacar que, en general, la utilidad principal de estos algoritmos está en encontrar representaciones buenas de la dinámica de transición del ambiente, ya que es en esta tarea donde más se puede explotar el supuesto de un MDP de bajo rango. Esto quiere decir que, en general, para resolver el problema de encontrar una política optimal, se deberá hacer un paso adicional (ej, Policy Iteration) pero usando las representaciones del ambiente encontradas. Ahora bien, al final del capítulo se verá un tipo de algoritmo que intenta resolver tanto el problema de encontrar una representación de la dinámica del ambiente como el de encontrar una política optimal, ambos de forma simultanea.

3.1 Algunos Supuestos del Problema

Lo primero antes de entrar a detalle con los diferentes algoritmos, será dejar en claro los supuestos que se manejarán y que logran hacer que estas distintas aproximaciones sean comparables.

1. **Dinámica de transición estacionaria:** Si bien el algoritmo descrito en [Agarwal et al., 2020] está originalmente pensado para una dinámica de transición $\mathbb{P}_h$ que va cambiando con cada paso $h \in [H]$ del MDP, este mismo se puede adaptar para una dinámica estacionaria, igual al caso descrito en 2.2. Además, note que todo MDP de horizonte finito con dinámica no estacionaria se puede transformar en un MDP estacionario, como en 2.2, al redefinir los estados $\mathcal{X} \to \mathcal{X}^H$ donde cada estado nuevo tendrá la información del paso en el que está. Es decir, se puede interpretar que los MDP con dinámicas no estacionarias en realidad están imponiendo una restricción sobre el espacio de estados $\mathcal{X}$.

2. **Función de recompensa conocida:** Es muy común en los problemas de RL que la función de recompensa sea diseñada por el usuario para que el agente aprenda a realizar algún objetivo en particular. Por ende, no es descabellado pensar que se tienen unas recompensas completamente conocidas y, de hecho, esto es justamente uno de los supuestos en los algoritmos que se verán.

3. **Espacio de estados y acciones finito:** El algoritmo descrito en [Uehara et al., 2021] es el único de los 3 diseñado para funcionar con espacios infinitos. Más aún, la única forma de adaptar [Zhu et al., 2021] al contexto de MDPs de bajo rango es teniendo espacios de acciones y estados finitos. Por ende, esta será la situación en la cual se enfocará este trabajo.

4. **Online RL:** Los algoritmos en [Agarwal et al., 2020] y [Uehara et al., 2021] son particularmente eficientes si tienen acceso a interacciones con el ambiente,

ya que logran explotar la estructura subyacente de bajo rango del problema al realizar una exploración inteligente que, mediante políticas exploratorias, logran maximizar la probabilidad de observar dichas estructuras de bajo rango.

5. **Simplex Feature MDP:** Por último, el supuesto de bajo rango, como se vió en 2.3 engloba gran cantidad de casos. Pero para la experimentación y para las simulaciones se usará en específico MDPs con simplex features, simplemente porque son más sencillos de simular.

3.2 Learning Markov Models Via Low-Rank Optimization

Previo a estudiar cómo resolver MDPs de bajo rango, una buena aproximación será la de primero intentar aproximar cadenas de Markov de bajo rango y luego hacer el salto a MDPs.

La aproximación de matrices de bajo rango es un problema clásico y bien estudiado en la literatura. En este trabajo nos concentraremos en los resultados de [Zhu et al., 2021], específicamente su primer algoritmo, el cual usa técnicas de optimización convexa para desarrollar un algoritmo que aproxime una matriz de transición de bajo rango de una cadena de Markov ergódica.

3.2.1 Supuestos y Notación de sGS-ADMM

En dicho trabajo, se tiene una cadena de Markov ergódica con matriz de transición real $P \in \mathbb{R}^{p\times p}$, donde p es el número de estados. Dicha matriz se presume de bajo rango con rango real r. Por otro lado, si $\mathcal{X}$ es el espacio de estados y $\mathcal{X} = (X_0, \ldots, X_n)$ una trayectoria de nuestra cadena de markov con $X_i \in \mathcal{X}$, se define n_{ij} como la suma de todas las veces que se transicionó del estado i al j en nuestra cadena de Markov $\mathcal{X}$, i.e,

$$n_{ij} = |\{1 \leq k \leq n : X_{k-1} = i \wedge X_k = j\},|$$

Con esto definimos la función de log-verosimilitud negativa promediada.

Definición 3.1 (Negative Log-Likelihood Function)**.** Dada una matriz $P \in \mathbb{R}^{p\times p}$ se define la función de log-verosimilitud negativa como:

$$\ell_n(P) = -\frac{1}{n}\sum_{i=1}^{p}\sum_{j=1}^{p} n_{ij}\ln(P_{ij}).$$

donde $n_i = \sum_{j=1}^{p} n_{ij}$ y $n = \sum_{i=1}^{p} n_i$. Una vez tenemos esta función el estimador propuesto en [Zhu et al., 2021] para aproximar la matriz de transición, es el estimador de máxima verosimilitud (MLE) regularizado con la norma nuclear, el cual se define como la solución del problema de optimización convexo:

$$\begin{aligned} \hat{P} &= \arg\min \ell_n(Q) + \lambda \parallel Q \parallel_* \\ s.a.\ & Q1_p = 1_p \quad \frac{\alpha}{p} \leq Q_{ij} \leq \frac{\beta}{p}, \end{aligned} \tag{1}$$

donde 1_p es el vector de unos de dimensión p, α y β son constantes no negativas y $\lambda > 0$ es un parámetro de penalización.

Teorema 3.1 (Garantías Estadísticas del Estimador Propuesto en (1))**.** Si P es la matriz de transición real y $\hat{P}$ nuestro estimador, entonces, existe una constante c tal que si tenemos n muestras de la cadena de Markov y $n \leq c \cdot (r \cdot p \log p)^2$ entonces:

$$\| \hat{P} - P \|_F^2 = O_{\mathbb{P}}\left(\frac{rp \log p}{n}\right)$$

En el artículo original destacan otras cotas estadísticas para n más grande, pero esta depende del rango exacto $[\frac{\alpha}{p}, \frac{\beta}{p}] \subseteq [0, 1]$ sobre el que varíen las entradas de P.

Nota: Para ver más sobre las garantías estadísticas de este estimador ver Teorema 1 en [Zhu et al., 2021].

El problema de calcular el MLE (1), tomando $\alpha = 0, \beta = p, g(X) = \ell_n(X) + \delta(X \geq 0), \mathcal{A}(X) = X1_p$ y $b = 1_p$, se puede ver como un caso particular de la familia de problemas de optimización:

$$\text{mín}\{g(X) + c \| X \|_* : \mathcal{A}(X) = b\} \tag{2}$$

donde $g : \mathbb{R}^{p \times p} \to (-\infty, \infty]$ es una función convexa y cerrada, $\mathcal{A} : \mathbb{R}^{p \times p} \to \mathbb{R}^m$ lineal, $b \in \mathbb{R}^m$ y $c > 0$.

A pesar de la convexidad del problema su solución es bastante no trivial, sobre todo por la falta de suavidad de la función g y la presencia de la norma nuclear. Por lo tanto, se recurre el la formulación dual de (1).

$$\text{mín } g^*(\Xi) - \langle b, y \rangle \tag{3}$$

$$s.a. \ \Xi + \mathcal{A}^*(y) + S = 0 \quad \| S \|_2 \leq c$$

donde $\| \cdot \|_2$ es la norma espectral y g^* es la función conjugada de g dada por

$$g^*(\Xi) = \sum_{(i,j) \in \Omega} \frac{n_{ij}}{n} (\ln \frac{n_{ij}}{n} - 1 - \ln(-\Xi_{ij})) + \delta(\Xi \leq 0),$$

con $\Omega = \{(i, j) : n_{ij} \neq 0\}$.

El Lagrangiano aumentado dado $\sigma > 0$ asociado al problema (3) es:

$$\begin{aligned} \mathcal{L}_\sigma(\Xi, y, S; X) = g^*(-\Xi) - \langle b, y \rangle - \frac{1}{2\sigma} \| X \|^2 \\ + \frac{\sigma}{2} \| \Xi + \mathcal{A}^*(y) + S + X/\sigma \|^2 \end{aligned} \tag{4}$$

Dado que hay tres bloques separables en (3), más específicamente Ξ, y y S, el ADMM directo extendido no es aplicable. Como se muestra en [Chen et al., 2016] ADMM directo extendido no es necesariamente convergente para problemas de minimización convexa con multibloques. Sin embargo, como las funciones correspondientes al bloque y son lineales es posible aplicar el método Gauss-Sediel simétrico multibloque basado en ADMM (sGS-ADMM) [Li et al., 2016]. Este algoritmo se expone en la sección siguiente.

3.2.2 sGS-ADMM Algoritmo

Aunque las expresiones para las iteraciones de Ξ, y, S, X en el algoritmo 3 parecen complicadas, en todos los casos se puede obtener una fórmula explícita de la k-ésima iteración. En esta sección se expondrá dichas soluciones cerradas, para una discusión más completa de cómo se llega a estas soluciones cerradas dirigirse al Apéndice B.

$$y^{k+1} = \frac{1}{\sigma}(\mathcal{A}\mathcal{A}^*)^{-1}(b - \mathcal{A}(X^k - \sigma(\Xi^{k+1} + S^k))) \tag{5}$$

$$y^{k+\frac{1}{2}} = \frac{1}{\sigma}(\mathcal{A}\mathcal{A}^*)^{-1}(b - \mathcal{A}(X^k - \sigma(\Xi^k + S^k))) \tag{6}$$

Además se puede probar que $\mathcal{A}\mathcal{A}^* y = py$.
Sea $R^k = \mathcal{A}^*(y^{k+\frac{1}{2}} + S^k + X^k/\sigma)$ y Z_{ij}^k tal que:

$$Z_{ij}^k = \begin{cases} \frac{\sigma R_{ij}^k + \sigma\sqrt{(R_{ij}^k + 4n_{ij}/(n\sigma))^2}}{2} & (i,j) \in \Omega \\ \sigma \,\text{máx}(R_{ij}^k, 0) & (i,j) \notin \Omega \end{cases}$$

Se puede demostrar que:

$$\Xi^{k+1} = \frac{1}{\sigma}(Z^k - \sigma R^k), \tag{7}$$

Por último, tenemos que la expresión explícita para S^{k+1} es:

$$S^{k+1} = U_k \,\text{mín}(\Sigma_k, c) V_k^t, \tag{8}$$

y $\text{mín}(\Sigma_k, c) = \text{Diag}(\text{mín}(\alpha_1, c), ..., \text{mín}(\alpha_n, c))$ con $\alpha_1 \geq ... \geq \alpha_n$ los valores singulares de una matriz W_k donde $W_k := U_k \Sigma_k V_k^t$.

Algorithm 3 sGS-ADMM

Require: Punto inicial (Ξ^0, y^0, S^0, X^0), un parámetro de penalización $\sigma > 0$, un máximo número de iteración K y una longitud de paso $\gamma \in (0, \frac{1+\sqrt{5}}{2})$.
for $k = 0$ **to** K **do**
$y^{k+\frac{1}{2}} = \arg\text{mín}_y \mathcal{L}_\sigma(\Xi^k, y, S^k; X^k)$
$\Xi^{k+1} = \arg\text{mín}_\Xi \mathcal{L}_\sigma(\Xi, y^{k+\frac{1}{2}}, S^k; X^k)$
$y^{k+1} = \arg\text{mín}_y \mathcal{L}_\sigma(\Xi^{k+1}, y, S^k; X^k)$
$S^{k+1} = \arg\text{mín}_S \mathcal{L}_\sigma(\Xi^{k+1}, y^{k+1}, S; X^k)$
$X^{k+1} = X^k + \gamma\sigma(\Xi^{k+1} + \mathcal{A}^*(y^{k+1}) + S^{k+1})$.
end for

3.2.3 Usando sGS-ADMM para resolver Low-Rank MDPs

La idea de este trabajo es adaptar el algoritmo 3 al contexto de un MDP de bajo rango. Para ello, note que si tenemos un MDP $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{R}, P)$ con espacios de acciones y estados finitos, cada acción $a \in \mathcal{A}$ induce una matriz $M_a \in \mathbb{R}^{|\mathcal{X}|\times|\mathcal{X}|}$ donde la entrada ij de M_a está dada por:

$$M_a(ij) := \mathbb{P}(x_i, a, x_j)$$

Donde x_i, x_j son el estado i y j asumiendo una numeración de $\mathcal{X}$, lo cual se puede hacer si este es finito. Es evidente que M_a es una matriz de transición. Si, además, suponemos que para toda política, las matrices de transición M_a son irreducibles, tendremos una matriz de transición ergódica sobre la cual podremos aplicar el algoritmo 3 para aproximarla. Esto tiene sentido ya que un MDP de bajo rango induce matrices M_a de bajo rango. Podemos notar esto con las siguientes observaciones:

Teorema 3.2. Sea $\mathcal{M}$ es un MDP de bajo rango con dinámica de transición dada por ϕ, μ y rango d. Sea $U \in \mathbb{R}^{d\times|\mathcal{X}|}$ tal que $U(ij) := \mu(x_j)[i]$, es decir, la entrada i-esima de $\mu(x_j)$. Por otro lado, sea $Ph_a \in \mathbb{R}^{|\mathcal{X}|\times d}$ tal que $Ph_a(ij) := \phi(x_i, a)[j]$, entonces, $M_a = Ph_a \cdot U$, lo que implica que el rango de M_a es d para todo $a \in \mathcal{A}$.

La demostración es simplemente seguir la definición de $\mathbb{P}$ como producto punto de ϕ y μ para luego ver que esto es exactamente lo mismo que multiplicar vectores fila de Ph por vectores columna de U. Entonces, el algoritmo propuesto recogerá $N = |\mathcal{A}| \cdot d|\mathcal{X}| \log |\mathcal{X}|$ muestras del MDP, usando una política exploratoria uniforme sobre las acciones, para luego hacer una iteración de sGS-ADMM para cada acción a con los datos de transición vistos para los cuales la acción tomada fue a. En general, cada iteración del algoritmo debería contar con aproximadamente $n_a = d|\mathcal{X}| \log |\mathcal{X}|$ muestras de transición debido a la política uniforme, lo que garantiza los resultados estadísticos del teorema 3.1. Correr este algoritmo crece de forma lineal con el número de acciones. Además, como se calcula cada M_a por separado, no se logra capturar la información de que todo M_a es el resultado de multiplicar una matriz Ph_a por la misma matriz U para todas las acciones, aquella que es inducida por μ.

Algorithm 4 Low-rank MDP based sGS-ADMM

Require: $D_a = \emptyset \ \forall a \in \mathcal{A}$. Un parámetro de penalización $\sigma > 0$, un máximo número de iteración K y una longitud de paso $\gamma \in (0, \frac{1+\sqrt{5}}{2})$. Además, $\pi = unif(\mathcal{A})$ y $N := |\mathcal{A}| \cdot d|\mathcal{X}| \log |\mathcal{X}|$

for $k = 0$ **to** N **do**
  Tome (x, a, x') con $a \sim \pi(x)$
  Agregue (x, a, x') a D_a.
end for
for $k = 0$ **to** $|\mathcal{A}|$ **do**
  $\hat{M}_a :=$ sGS-ADMM con D_a, σ, γ dados. El punto inicial será el MLE del M_a real dado por los datos de transición en D_a.
end for

3.3 FLAMBE

FLAMBE es un algoritmo diseñado por investigadores de Microsoft Research el cual da una solución probablemente polinomial en datos a cierto tipo de problemas de RL.

En específico, se concentra en un low rank MDP episódico, con episodios de largo H, con un espacio de acciones finito $\mathcal{A} = \{1, \ldots, K\}$ y una dinámica de transición no estacionaria. Usaremos la misma notación que en [Agarwal et al., 2020] y, para $H \in \mathbb{N}$, denotaremos al conjunto $\{1, \ldots, H-1\}$ por $[H]$ Esto quiere decir, que las probabilidades de transición entre estados, dada una acción tomada se asume que depende del tiempo actual. Por ende, para cada $h \in [H]$ tendremos una función $\mathbb{P}_h$ que describe las dinámicas de transición en dicho paso.

Luego, el algoritmo se basa en dos ideas importantes: aproximar las funciones ϕ_h y μ_h usando clases de funciones dadas por el usuario y luego explorar el ambiente de forma inteligente para que los datos observados tengan alta probabilidad de describir la dinámica de bajo rango del ambiente. Note que como estamos suponiendo una dinámica de transición no estacionaria debemos aproximar cada $\mathbb{P}_h$ con dos funciones ϕ_h, μ_h.

Muchos de los resultados teóricos de FLAMBE son altamente dependientes del tipo de funciones que se usen para aproximar la dinámica real del sistema, la cual suponen está dada por $\mathbb{P}_h(x, a, x') = \langle \hat{\phi}_h(x, a), \hat{\mu}_h(x') \rangle$, entonces, el objetivo del algoritmo es encontrar otras funciones $\phi_h \in \Phi$ y $\mu_h \in \Upsilon$ que mejor aproximen las originales, donde Φ, Υ son las clases dadas por el usuario.

3.3.1 Supuestos de FLAMBE

El algoritmo desarrollado en [Agarwal et al., 2020] tienen una serie de supuestos concernientes a la naturaleza del problema a resolver. Algunos de los supuestos más fuertes consisten en la existencia de llamados *Oráculos* computacionales que permiten calcular ciertos valores de forma eficiente. Otros de los supuestos hablan sobre la naturaleza de los modelos Φ, Υ usados sobre los cuales se quiere encontrar las aproximaciones ϕ_H y μ_h de la dinámica real. En general gran parte de los resultados teóricos de complejidad presentados en [Agarwal et al., 2020] asumen que Φ y Υ son espacios finitos. En la práctica, cualquier espacio decente sobre el cual buscar funciones será infinito y lo que se querrá en realidad es que se pueda resolver (incluso de forma aproximada) el problema de encontrar el MLE en estos espacios. Concerniente a dicho requerimiento, tanto [Agarwal et al., 2020] como [Uehara et al., 2021] definen el siguiente oráculo:

- **El Oráculo de Máxima Verosimilitud (MLE):** Dado un conjunto de datos $\mathcal{D}$ que contiene triplas observadas de transiciones de estados (x_i, a_i, x_{i+1}), este oráculo puede obtener de forma eficiente los elementos $\phi \in \Phi, \mu \in \Upsilon$ tales que:

$$(\phi, \mu) := \underset{\phi \in \Phi, \mu \in \Upsilon}{argmax} \sum_{(x,a,x') \in \mathcal{D}} \log \langle \phi(x, a), \mu(x') \rangle$$

En la sección 3.3.5 se discutirá un poco más a fondo del tipo de clases de funciones que se usarán como Φ y Υ, que sean lo suficientemente expresivas como para encontrar buenas aproximaciones y, además, que permitan optimizar de forma eficiente para acercarse al **MLE** con costo computacional relativamente bajo.

En esta misma linea, otro de los supuestos teóricos de FLAMBE tiene que ver con *realizability* y *reachability*.

3.3.1.1 Realizability Assumption

Esto supone que las funciones reales $\hat{\phi}_h$ y $\hat{\mu}_h$ pertenecen a Φ y Υ respectivamente. Esto les interesa ya que garantiza que en algún punto, si Φ y Υ son finitos, se alcanzará a llegar a $\hat{\phi}_h$ y $\hat{\mu}_h$ con alta probabilidad.

3.3.1.2 Reachability Assumption

Por otro lado, *reachability* presupone que existe una política π que, al seguirla, se puede pasar por todos los estados latentes del modelos con probabilidad mayor a 0, es decir, si $z_i \in \mathcal{Z}$ es un estado latente de mi MDP $\mathcal{M}$, entonces, dado un tiempo $h \in [H]$ y un estado $x \in \mathcal{X}$, existe acción $a \in \mathcal{A}$ tal que la i-esima entrada de $\hat{\phi}_h(x, a)$ es mayor a 0. Esto tiene un paralelo directo con la suposición del capítulo anterior mediante el cual todas las matrices de transición M_a inducidas eran ergódicas.

El segundo método que usa FLAMBE para explotar el supuesto de bajo rango tiene que ver con la forma en la que explora el ambiente. Estos algoritmos (se verán más a fondo en la sección 3.3.3) dependen de la existencia de otro oráculo computacional:

- **Orácculo de muestreo (SAMP):** Dadas funciones $\phi \in \Phi, \mu \in \Upsilon$ y $x \in \mathcal{X}, a \in \mathcal{A}$, el orácculo **SAMP** permite devolver de forma eficiente una muestra $x' \in \mathcal{X}$ siguiendo la distribución de $\langle \phi(x, a), \mu(\cdot) \rangle \in \Delta(\mathcal{X})$. Es decir, esto es simplemente una subrutina eficiente de muestreo.

3.3.2 El Algoritmo

El algoritmo 5 es el original y resultado principal del paper [Agarwal et al., 2020]. Se puede observar que el mecanismo base del algoritmo es bastante simple. Solo se está calculando el **MLE** de forma iterativa basado en observaciones que se almacenan en D_h. Este proceso se repite J_{max} veces para obtener los resultados obtenidos en dicho paper. Este valor debe ser el siguiente:

$$J_{max} := \frac{4Hd}{K\sqrt{\varepsilon_{TV}}} \cdot \log\left(1 + \frac{4H}{K\sqrt{\varepsilon_{TV}}}\right), \tag{9}$$

donde d es la dimensión de nuestro espacio latente que, en el caso de simplex features, es justamente igual a d_{LV}, es decir, al número de estados latentes (recuerde que ϕ va de $\mathcal{X} \times \mathcal{A}$ a $\mathbb{R}^d$). Además $K = |\mathcal{A}|$, es decir, el número de acciones posibles. Por último, ε_{TV} sirve para establecer una cota superior para la variación total entre el MDP aproximado y el real, tal que se cumple la siguiente desigualdad:

$$\forall x \in \mathcal{X}, \forall a \in \mathcal{A} : \mathbb{E}\Big[\parallel \mathbb{P}_1(\cdot|x, a) - \hat{\mathbb{P}}_1(\cdot|x, a) \parallel_{TV} \Big] \leq K\sqrt{\varepsilon_{TV}} \tag{10}$$

donde $\mathbb{P}_1$ es la dinámica verdadera en el tiempo 1 y $\hat{\mathbb{P}}_1$ la dinámica aproximada por el algoritmo. Claramente en la práctica este valor no se conoce y deberá ser escogido.

Algorithm 5 FLAMBE para dinámica no estacionaria. (Original presentado en [Agarwal et al., 2020])

Require: Ambiente $\mathcal{M}$, clases de funciones Φ, Υ óraculus MLE y SAMP, parámetros β, n, ρ_0 la política que escoge acciones uniformemente sobre el espacio de acciones.
Sea $D_h = \emptyset$ para $h \in \{0, \dots, H-1\}$
for $j = 1, \dots, J_{max}$ **do**
  for $h = 0, \dots, H-1$ **do**
    Tome n observaciones de triplas (x_h, a_h, x_{h+1}) llegando a x_h siguiendo ρ_{h-1} y tome $a_h \sim unif(\mathcal{A})$.
    Luego, añada (x_h, a_h, x_{h+1}) a D_h.
    Encontrar MLE: $(\hat{\phi}_h, \hat{\mu}_h) := \underset{\phi \in \Phi, \mu \in \Upsilon}{argmax} \sum_{D_h} \log \langle \phi(x, a), \mu(x') \rangle$
  end for
  Para cada h llamar al planner dado por el algoritmo 7 o el 8 con los modelos aprendidos $(\hat{\phi}_{0:h-1}, \hat{\mu}_{0:h-1})$ y el parámetro β para obtener ρ_h^{pre}. Y asigne $\rho_j := unif(\{\rho_h^{pre} \circ random\}_{h=0}^{H-1})$
end for

El parámetro β es requerido por la versión de FLAMBE que usa el algoritmo 7 como planner, pero si se está en una situación de simplex features donde se pueda usar 8, este valor no es requerido. Más adelante se verá por qué β debe ser menor a 1 para que el planner 7 funcione correctamente.

Es claro que el algoritmo 5 necesita un pequeño ajuste para el caso con dinámicas estacionarias. En este caso tendremos que $\mathbb{P}_i = \mathbb{P}_j$, por ende, el MLE se puede calcular sobre todas las muestras obtenidas no solamente sobre las transiciones del paso h al $h+1$. El algoritmo 6 muestra dichos cambios.

Algorithm 6 FLAMBE para dinámica estacionaria

Require: Ambiente $\mathcal{M}$, clases de funciones Φ, Υ óraculus MLE y SAMP, parámetros β, n, ρ_0 la política que escoge acciones uniformemente sobre el espacio de acciones.
Sea $D = \emptyset$.
for $j = 1, \dots, J_{max}$ **do**
  Tome n observaciones de triplas (x_j, a_j, x_{j+1}) llegando a x_j siguiendo ρ_{j-1} y tome $a_j \sim unif(\mathcal{A})$.
  Luego, añada (x_j, a_j, x_{j+1}) a D.
  Encontrar MLE: $(\hat{\phi}, \hat{\mu}) := \underset{\phi \in \Phi, \mu \in \Upsilon}{argmax} \sum_{D} \log \langle \phi(x, a), \mu(x') \rangle$
  Llamar al planner dado por el algoritmo 7 o el 8 con los modelos aprendidos $(\hat{\phi}, \hat{\mu})$, $h = 1$ y el parámetro β para obtener ρ^{pre}. Y asigne $\rho_j := unif(\{\rho^{pre} \circ random\})$
end for

Hasta el momento se ha pasado un poco por alto la utilidad e intuición detrás de los algoritmos de planeación que usa FLAMBE. En la sección 3.3.3 se hablará de dichas técnicas para explorar eficientemente.

3.3.3 Algoritmos de Planeación

Cómo ya se había mencionado, FLAMBE explota el supuesto de bajo rango del MDP de 2 formas, la primera ya se vió anteriormente y es simplemente representando la dinámica del sistema con modelos Φ y Υ que capturarán el mecanismo de bajo rango. Por otro lado, el segundo método aprovecha la existencia de un oráculo de muestreo (SAMP) y del online RL para intentar que al momento de explorar el ambiente, la probabilidad de observar la estructura latente de bajo rango sea lo más alta posible, esto es, intenta construir una política exploratoria que cubra todas o la mayoría de direcciones correspondientes a los estados latentes.

3.3.3.1 Elliptical Planner

El planeador genérico para low-rank MDPs que usa FLAMBE es el algoritmo 7, pero tanto este como el algoritmo 8, el cual es específico para el caso de simplex features, se centran en la función aprendida $\hat{\phi}$, la cual nos da una representación d-dimensional del espacio latente $\mathcal{Z}$ que en el caso de simplex features coincide con probabilidades de transición a las variables latentes. En general, esto último no siempre se tendría que cumplir.

La idea entonces es que el algoritmo planeador nos permita observas muchas direcciones sobre mi espacio latente tal que cubran en gran medida todo el espacio.

Algorithm 7 Elliptical Planner. Algoritmo 2 de [Agarwal et al., 2020]

Require: Ambiente $\tilde{\mathcal{M}} := (\phi_{0:\hat{h}-1}, \mu_{0:\hat{h}-1})$, oráculo SAMP y parámetro $\beta > 0$. Inicializar $\Sigma_0 = I_{d\times d}$

for $t = 1, \ldots$ **do**

Calcule:

$$\pi_t := \underset{\pi}{\operatorname{argmax}}\ \mathbb{E}[\phi_{\hat{h}}(x_{\tilde{h}}, a_{\tilde{h}})^T \Sigma_{t-1}^{-1} \phi_{\hat{h}}(x_{\tilde{h}}, a_{\tilde{h}}) | \pi, \tilde{\mathcal{M}}]$$

Si el valor máximo alcanzado en el paso anterior es menor o igual a β parar y retornar $\rho := unif(\{\pi_\tau\}_{\tau<t})$.

Calcular $\Sigma_{\pi_t} = \mathbb{E}[\phi_{\hat{h}}(x_{\tilde{h}}, a_{\tilde{h}}) \phi_{\hat{h}}(x_{\tilde{h}}, a_{\tilde{h}})^T | \pi, \tilde{\mathcal{M}}]$ y asigne $\Sigma_t := \Sigma_{t-1} + \Sigma_{\pi_t}$

end for

Para dar una intuición al respecto considere la primera iteración del algoritmo 7, para la cual la matriz de covarianzas empírica aún es la identidad ($\Sigma_0 = I_{d\times d}$). En esta iteración la política encontrada π_0 es tal que se maximice el valor esperado de la norma cuadrada de $\phi(x_1, a_1)$, pero recuerde que $\phi(x_1, a_1)$ pertenece a la bola unitaria d-dimensional, por ende, su norma se maximizará al estar cerca de las esquinas del simplex contenido en dicha bola, es decir, cuando una de las entradas se acerque a 1 y el resto se acerquen a 0. En el contexto de simplex features, esto quiere decir que el algoritmo encuentra una política que maximiza la probabilidad de llegar a uno de los estados latentes, en otras palabras, encuentra una dirección de exploración. Para el caso más general, podemos interpretarlo como que está encontrado una dirección con alta importancia dentro de la dinámica del sistema. Lo ideal intuitivamente en siguientes iteraciones sería que el algoritmo fuera explorando direcciones importantes pero que aún no ha explorado, justamente esto hacen los planners.

Con el objetivo de analizar qué hace el algoritmo en siguientes iteraciones vale la pena tener presente cómo se calcula Σ_t^{-1}. Usualmente calcular inversas de matrices es un proceso costoso, pero, en este caso, se cuenta con la suerte que existe una fórmula cerrada para calcular la inversa de la matriz $\Sigma_t := I + \sum_{k=1}^{t-1} \mathbb{E}[\phi(x_1, a_1) \cdot \phi(x_1, a_1)^T | \pi_k]$. Estamos hablando de la fórmula de *Sherman-Morrison* la cual se define de la siguiente manera:

Teorema 3.3 (Fórmula de Sherman-Morrison [Sherman and Morrison, 1950])**.** Dados vectores $v_j \in \mathbb{R}^d, 0 < j \leq k$, I la identidad $d \times d$ y $\alpha > 0$. Definimos $S_k := \alpha I + \sum_{i=1}^{k} v_i v_i^T$. Entonces, se puede encontrar iterativamente la inversa de S_k de la siguiente forma:

$$S_k^{-1} = S_{k-1}^{-1} - \frac{w_k w_k^T}{1 + v_k^T w_k}, w_k := S_{k-1}^{-1} v_k \tag{11}$$

Para el algoritmo 7, se tendría que $S_k = \Sigma_k$, $\alpha = 1$ y $v_k = \mathbb{E}[\phi(x_1, a_1) | \pi_{k-1}]$. Entonces, para la $k-$esima iteración del algoritmo, si $v_k = \mathbb{E}[\phi(x_1, a_1) | \pi_{k-1}]$ y Σ_k^{-1} calculado como en el teorema 3.3 entonces el objetivo de maximización para π_k sería:

$$v_k^T \Sigma_k^{-1} v_k = v_k^T I v_k - \sum_{t=1}^{k} \frac{v_k^T (\Sigma_{t-1}^{-1} v_t)(\Sigma_{t-1}^{-1} v_t)^T v_k}{1 + v_t \Sigma_{t-1}^{-1} v_t} \tag{12}$$

Por lo que, intuitivamente, debemos de intentar maximizar $v_k^T I v_k$ mientras que simultáneamente minimizamos la resta. Es decir, debemos encontrar un v_k que vuelva grande $\langle v_k, v_k \rangle$ y pequeño $\langle \Sigma_{t-1}^{-1} v_t, v_k \rangle$ para todo t. La primera condición fuerza al modelo a encontrar un v_k cercano a la esfera unitaria, en particular, en el caso del simplex lo lleva a buscar direcciones cercanas a las esquinas del simplex, igual que en el primer paso. Para la segunda condición, vale la pena recordar que los v_t se obtienen respectivamente maximizando $\langle \Sigma_{t-1}^{-1} v_t, v_t \rangle$. Si fijamos v_t, el producto punto se maximiza en las direcciones paralelas a v_t, por ende, podemos esperar que $\Sigma_{t-1}^{-1} v_t$ esté en una dirección relativamente similar a la dirección de v_t, por lo que se puede considerar la minimización de $\langle \Sigma_{t-1}^{-1} v_t, v_k \rangle$ como algo cercano a minimizar $\langle v_t, v_k \rangle$, es decir, de ir en una dirección opuesta a v_t. En el caso de simplex features, sería ir en una dirección ortogonal a v_t porque aquí todos los valores serían positivos y solo se podría minimizar el producto punto acercándolo a 0. En general se está buscando una dirección simultaneamente contraria a todas las previamente observadas y que tenga una norma cercana a 1.

Para ver cuándo debería parar el algoritmo, pensemos de nuevo en la tarea de maximizar $\langle \Sigma_{t-1}^{-1} v_t, v_t \rangle$. Este valor va a estar siempre acotado por $\langle c \cdot v_t, v_t \rangle$ para algún c, por ende, el v_t que maximice dicho producto será tal que $\Sigma_{t-1}^{-1} v_t$ estará cerca de $c \cdot v_t$ para algún c o, en otras palabras, el algoritmo buscará algún v_t cerca de los vectores propios de la matriz Σ_{t-1}^{-1} y lo más cerca a tener norma 1 que pueda. La cota superior a este problema sería el vector propio unitario de la matriz Σ_{t-1}^{-1} que corresponda al valor propio más grande. Claramente no es seguro que siquiera se pueda llegar cerca de dicha dirección con el valor esperado de la función ϕ, por ende, el algoritmo establece un parámetro β el cual será el que pare el ciclo. La idea detrás de este parámetro es que, en el momento en que $\langle \Sigma_{t-1}^{-1} v_t, v_t \rangle$ no se pueda hacer más grande, entonces quiere decir que las direcciones de exploración restantes que se pueden alcanzar con ϕ no

son lo suficientemente lejanas a las previamente exploradas por lo que el algoritmo deberá parar. Evidentemente el valor de β deberá estar siempre en $[0, 1]$, aunque muy seguramente debería estar más cercano a 0 que a 1 para lograr explorar mucho más el ambiente.

3.3.3.2 Simplex Features Planner

Este planner está pensado específicamente para el caso en el que el MDP tenga simplex features. El funcionamiento básico es muy similar a lo que hace el algoritmo 7 pero la estructura adicional del MDP permite que sea mucho más sencillo este método que el usado en el planeador elíptico. De nuevo se buscan direcciones importantes y cercanas a tener norma 1, pero en este caso solo se buscan dentro del simplex, ya que allí vive ϕ.

Con esto en mente, lo más sencillo de hacer es encontrar la manera de que la política exploratoria haga que ϕ se acerque a las esquinas del simplex, o, dicho de otra manera, queremos que las políticas exploratorias maximicen una sola de las d entradas de ϕ. Entonces se hace este proceso para cada una de las d entradas de ϕ y se devuelve una política uniforme sobre el conjunto de políticas encontradas donde cada una maximiza una entrada específica de ϕ. Al final, la política retornada siempre explorará sobre direcciones cercanas a las esquinas del simplex, aumentando la probabilidad de observar la dinámica de bajo rango con simplex features del sistema.

Algorithm 8 Planner for Simplex MDP. Algoritmo 3 de [Agarwal et al., 2020]

Require: Ambiente $\tilde{\mathcal{M}} := (\phi_{0:\hat{h}-1}, \mu_{0:\hat{h}-1})$. Sea d_{LV} el número estimado de estados latentes:
for $z = 1, \ldots, d_{LV}$ **do**
Calcule:

$$\pi_z := \underset{\pi}{\operatorname{argmax}}\ \mathbb{E}[\phi_{H-1}(x_{H-1}, a_{H-1})[z] | \pi, \tilde{\mathcal{M}}]$$

donde $\phi(x, a)[i]$ es la i-esima entrada del vector d_{LV} dimensional dado por $\phi(x, a)$
end for
Retornar $\rho := unif(\{\pi_z\}_{z=1}^{d_{LV}})$.

3.3.4 Principales Resultados Teóricos de FLAMBE

Los resultados teóricos de este algoritmo se enmarcan dentro de un objetivo propuesto por los autores, el cual tiene que ver con aproximar de una manera uniforme la dinámica real del sistema sin tener en cuenta la señal de recompensa.

Definición 3.2 (Objetivo de Aprendizaje)**.** Dadas clases de funciones Φ y Υ, el algoritmo debe aprender funciones $\hat{\phi} \in \Phi$ y $\hat{\mu} \in \Upsilon$ tales que:

$$\forall \pi, x, a : \mathbb{E}[\| \langle \hat{\phi}(x, a), \hat{\mu}(\cdot) \rangle - \mathbb{P}(\cdot | x, a) \|_{TV} | \pi, \mathcal{M}] \leq \varepsilon, \tag{13}$$

para algún $\varepsilon > 0$. Esto implica que el modelo aprendido aproxima bien la dinámica de transición inducida por seguir una política π para cualquier política posible.

Al cumplir con la definición anterior nos aseguramos que con el modelo aprendido podremos generalizar sobre todos los estados y, por ende, será posible usar dicho modelo con algoritmos estándar de RL como value iteration para encontrar políticas que sean probablemente óptimas. Más específicamente, si se cumple (13), tenemos el siguiente resultado que nos permite usar técnicas tradicionales de programación dinámica.

Lemma 3.4 (Lemma 1)**.** Si $\hat{\mathcal{M}} = (\hat{\phi}, \hat{\mu})$ satisface el objetivo de aprendizaje (13), entonces:

$$\forall V : \mathcal{X} \to [0,1], \exists \theta : max_{\pi} \mathbb{E}\Big[\Big|\langle \theta, \hat{\phi}(x_1, a_1)\rangle - \mathbb{E}[V(x_2)|x_1, a_1]\Big||\pi, \mathcal{M}\Big] \leq \varepsilon \qquad (14)$$

Teorema 3.5 (Theorem 2)**.** Dado $\delta \in (0,1)$. Si $\mathcal{M}$ es un MDP de bajo rango con dimensión d y horizonte H, con el supuesto (3.3.1.1), entonces FLAMBE con el planner 7 obtiene un modelo estimado $\hat{\mathcal{M}}$ tal que (14) se cumple con probabilidad $1-\delta$ y el número total de trayectorias obtenidas es:

$$O\Big(\frac{H^{22}K^9d^7 \log\left(|\Phi||\Upsilon|/\delta\right)}{\varepsilon^{10}}\Big)$$

Y el algoritmo corre en tiempo polinomial un número polinomial de llamadas a los oráculos MLE y SAMP.

Teorema 3.6 (Theorem 3)**.** Dado $\delta \in (0,1)$. Si $\mathcal{M}$ es un MDP de bajo rango con *Simplex Features* y dimensión de espacio latente d_{LV}, se satisface el supuesto ((3.3.1.1)), y para todo $\phi \in \Phi$ se cunmple $\phi(x,a) \in \Delta(\{0,\ldots,d_{LV}\})$, entonces FlAMBE con el algoritmo de planeación 8, calcula un modelo estimado $\hat{\mathcal{M}}$ tal que (14) se cumple con probabilidad $1-\delta$ y el número total de trayectorias obtenidas es:

$$O\Big(\frac{H^{11}K^5d_{LV}^5 \log\left(|\Phi||\Upsilon|/\delta\right)}{\varepsilon^{3}}\Big)$$

Y el algoritmo corre en tiempo polinomial un número polinomial de llamadas a los oráculos MLE y SAMP.

En general, es evidente que mientras ε sea más pequeño, mejor estaremos aproximando nuestra dinámica real, pero, del mismo modo, cuento más se acerque a cero, crecerá de forma exponencial el número de trayectorias que tiene que obtener el modelo según lo establecido en los teoremas 3.5 y 3.6. Además, vemos que la complejidad escala de forma polinomial con el tamaño de las clases Φ y Υ, aunque en la práctica, como se ha dicho antes, lo que en realidad se quiere es que las clases de funciones Φ y Υ permitan optimizar de algún modo sobre ellas de forma eficiente y que sean lo suficientemente expresivas para aproximar las funciones reales. Por ejemplo, se podría usar redes neuronales.

3.3.5 Aproximadores de Funciones

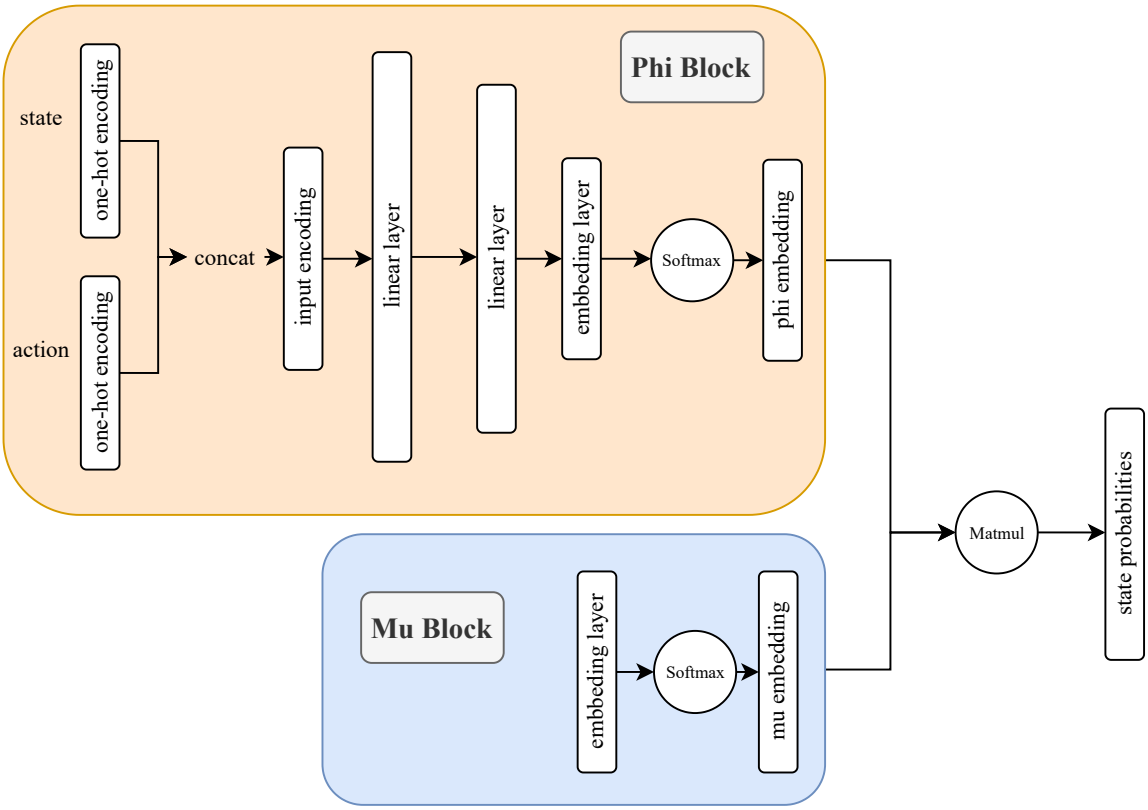


Figura 3.1: Arquitectura propuesta para una red neuronal que aproxime la dinámica de transición de un MDP de bajo rango

Para implementar tanto FLAMBE como REP-UCB se necesitan clases de funciones Φ y Υ sobre las cuales se pueda optimizar de forma relativamente eficiente y que sean lo suficientemente expresivas para aproximar de buena manera el MDP real. En este sentido, proponemos usar una arquitectura de redes neuronales que capture la información pertinente a un MDP de bajo rango para luego optimizar sobre ella usando el conocido algoritmo de backpropagation. En la figura 3.1 se muestra un diagrama de la arquitectura propuesta. Esta consta de dos bloques que harán las veces de nuestras clases de funciones. La idea general es que como input se reciben un estado y una acción, luego se ingresa a la red la concatenación de los vectores obtenidos al hacer un *one-hot encoding* del estado y la acción. Luego el bloque **Phi** producirá un vector resultante con un tamaño correspondiente al rango de nuestro MDP. Este bloque es directamente la clase Φ. Por otro lado, el bloque **Mu** contiene una matriz de parámetros de tamaño $d \times |\mathcal{X}|$, la cual tendrá la i-esima columna el vector correspondiente a $\mu(x_i)$ donde x_i es el i-esimo estado para alguna numeración fija del espacio de estados. Al final la multiplicación entre el bloque **Phi** y el bloque **Mu** dará por resultado un vector del tamaño del espacio de estados con las probabilidades estimadas de pasar a cada estado. La función de costo de esta red será precisamente la *Negative Log Likelihood*, lo que implica que minimizar esta función es equivalente a hallar el MLE del espacio de funciones dados los datos de entrenamiento. Todo esto es fácilmente implementado y entrenado en un framework de deeplearning como Pytorch y este código y el de los

demás algoritmos, estará recogido en un repositorio de código abierto en el siguiente link.

3.4 REP-UCB

El algoritmo REP-UCB desarrollado en [Uehara et al., 2021] es una variación de FLAMBE que mejora las cotas estadísticas de FLAMBE junto con sus complejidades mientras que intenta resolver el problema de RL de encontrar una política optimal al mismo tiempo que resuelve el problema de intentar encontrar una aproximación de la dinámica del sistema. Este algoritmo logra hacer esto ya que la misma política exploratoria que usa es la que devuelve al finalizar como política optimal, por ende, no necesita un paso adicional de Value-Iteration. Pero entonces, como no puede concentrarse solo en explorar el ambiente, establece un bonus de exploración que se le agrega a la función original de recompensa para así balancear entre exploración del ambiente y explotación del modelo aprendido.

3.4.1 Supuestos de REP-UCB

Los supuestos de este algoritmo son prácticamente idénticos a los de FLAMBE, desde los oráculos computacionales, pasando por *reachability* y *realizability*, hasta las clases de funciones Φ y Υ. Lo único importante que cambia es que supone dinámicas de transición estacionaria, por lo que no necesita aprender un ϕ y un μ para cada paso del modelo. En este sentido, es más comparable con la versión de FLAMBE descrita en 6, que tiene en cuenta justo este caso. Por otro lado, REP-UCB no hace uso de subrutinas de planeación como en FLAMBE, pero el bonos de exploración toma casi que la misma forma que la descrita en el algoritmo de planeación 7, en donde se quería maximizar cierta forma cuadrática, ya que un valor alto de dicha forma cuadrática representaba que aún existían direcciones sin explorar en el modelo. En este caso se codifica dicha información en el bono de exploración en vez de hacerlo directamente generando políticas mediante un planner. En siguientes secciones se muestran los resultados teóricos expuestos en [Uehara et al., 2021] sobre el algoritmo REP-UCB. Para mayor detalle sobre la demostración de los resultados, el lector se puede dirigir a los apéndices A y B de [Uehara et al., 2021].

3.4.2 El Algoritmo

Algorithm 9 UCB-driven representation learning, exploration, and exploitation (REP-UCB). Algoritmo 1 de [Uehara et al., 2021]

Require: Modelo del ambiente $\mathcal{M} = \{(\phi, \mu) : \phi \in \Phi, \mu \in \Upsilon\}$. Parámetros regularizadores λ_n y α_n. Número de iteraciones N. Inicialice $D_0 = \emptyset$, $D'_0 = \emptyset$ y $\pi_0 = unif(\mathcal{A})$.
for episodios $n = 1, \ldots, N$ **do**
Tome observaciones de una tupla $(x, a, x', a', \hat{x})$ tal que:

$$x \sim d_{\mathbb{P}}^{\pi_{n-1}}, a \sim unif(\mathcal{A}), x' \sim \mathbb{P}(\cdot|x, a), a' \sim unif(\mathcal{A}), \hat{x} \sim \mathbb{P}(\cdot|x', a')$$

donde $d_{\mathbb{P}}^{\pi_{n-1}}$ es simplemente la distribución sobre los estados generada por el ambiente al seguir la política π_{n-1} en la iteración inmediatamente anterior.

Luego, añada (x, a, x') a D_n y $(x', a', \hat{x})$ a D'_n de forma que:

$$D_n := D_{n-1} + (x, a, x'); D'_n := D'_{n-1} + (x', a', \hat{x})$$

Encontrar MLE: $\hat{P}_n := (\hat{\phi}_n, \hat{\mu}_n) = \underset{\phi \in \Phi, \mu \in \Upsilon}{argmax} \sum_{D_n + D'_n} \log \langle \phi(x, a), \mu(x') \rangle$
Actualizar matriz de covarianza empírica: $\hat{\Sigma}_n := \lambda_n I + \sum_{x,a \in D_n} \hat{\phi}_n(x, a) \hat{\phi}_n(x, a)^T$
Definir bono de exploración:

$$\hat{b}_n(x, a) := min(\alpha_n \sqrt{\hat{\phi}_n(x, a)^T \hat{\Sigma}_n^{-1} \hat{\phi}_n(x, a)}, 2)$$

Por último, actualice la política, tal que: $\pi_n := argmax_\pi V^\pi_{\hat{P}_n, r + \hat{b}_n}$
Donde $V^\pi_{\hat{P}_n, r + \hat{b}_n}$ es la Value Function calculada siguiendo la política π sobre el ambiente $\hat{P}_n$ y sumándole el bono de exploración a la función de recompensa r.
end for

3.4.3 Resultados Principales de REP-UCB

Con el algoritmo descrito previamente se establecen las siguiente cotas estadíticas y de complejidad sobre el modelo y la política optimal aprendidas.

Teorema 3.7 (Cota PAC para REP-UCB (Teorema 4 en [Uehara et al., 2021]))**.** Fije $\delta \in (0, 1)$. Si $\varepsilon \in (0, 1)$. Sea $\hat{\pi} := uniform(\{\pi_1, \ldots, \pi_N\})$ y $\tilde{\pi} := argmax_\pi V^\pi_{P,r}$ donde P es la dinámica real del MDP y r la función de recompensa. Entonces, fije los siguientes parámetros:

$$\alpha_n = O\big(\sqrt{(|\mathcal{A}| + d^2)\gamma \log(|\mathcal{M}|n/\delta)}\big), \qquad \lambda_n = O\big(d \log(|\mathcal{M}|n/\delta)\big)$$

Entonces, con probabilidad $1 - \delta$, si $\hat{P}$ es la dinámica aprendida por REP-UCB, se tiene que:

$$V_{P,r}^{\tilde{\pi}} - V_{\hat{P},r}^{\hat{\pi}} \leq \varepsilon \tag{15}$$

Además, el número de muestras que el algoritmo necesitan son:

$$n = O\left(\frac{d^4|\mathcal{A}|^2 \log\left(|\mathcal{M}|/\delta\right)}{(1-\gamma)^5\varepsilon^2} \cdot \nu\right) \tag{16}$$

Donde ν solo tiene términos logarítmicos y su dependencia en $\mathcal{M}$ es a lo mucho $\log\left(\log\left(\mathcal{M}\right)\right)$. En el apéndice B se especifica el valor exacto de ν como es descrito en [Uehara et al., 2021].

En otras palabras, el Value-Function resultande de seguir la política aprendida por REP-UCB sobre el modelo aprendido, está cerca del Value-Function de una política optimal sobre la dinámica real. Claramente mientras más cercano a 0 queramos que sea esta diferencia más trayectorias deberá explorar el modelo, pero en este caso esto solo crece como el cuadrado de ε, mientras que en FLAMBE dicho exponente era 3 para *Simplex Features* y 10 en general. Por lo que es una mejora considerable en la complejidad del algoritmo resultante.

En el siguiente capítulo, entraremos de lleno en la implementación de los algoritmos previamente descritos y sus resultados prácticos.

4 Resultados

Para verificar los resultados teóricos obtenidos, lo primero que se necesita es simular un MDP que cumpla con las características de bajo rango. Esto puede no ser tan fácil en términos generales, pero para el caso de un MDP con *Simplex Features* es relativamente sencillo, por ende, nos concentramos en dicho caso.

4.1 Simulación de un MDP de Bajo Rango

Para simular un MDP $\mathcal{M}$ de bajo rango con *Simplex Features* que tenga p estados, k acciones y rango d lo que se hizo fue tomar una matriz $\hat{U} \in \mathbb{R}^{d\times p}$ con entradas aleatorias, luego, a dicha matriz se le aplicó una *softmax* por filas y se obtuvo $U \in \mathbb{R}^{d\times p}$, para la cual los vectores fila estarán en el simplex Δ^p. Además de esto, se fijó un tensor aleatorio $\hat{M} \in \mathbb{R}^{d\times p\times k}$ y luego se le aplicó una *softmax* sobre sobre la primera dimensión, de tal forma que al fijar la segunda y tercera dimensión (estados y acciones respectivamente), el vector d-dimensional resultante esté en el simplex Δ^d. Entonces, con esta construcción hecha, si se da un estado x y una acción a, denotamos $M[:, x, a]$ al vector d-dimensional obtenido al fijar la segunda dimensión de M por x y la tercera por a, la multiplicación $M[:, x, a]^T \cdot U$ será una lista de tamaño p que corresponde a probabilidades sobre el espacio de estados, por ende, solo basta samplear un estado siguiendo esta distribución discreta. En esta construcción, $\phi(x, a) = M[:, x, a]$ y $\mu(x) = U[:, x]$.

Sumado a esto, se puede fijar un estado inicial o fijar una distribución, por ejemplo, uniforme y si se quiere establecer un horizonte finito fijo de H pasos, esto es equivalente a decir que, si se han recorrido H pasos, el siguiente estado sea el estado inicial o distribuido de acuerdo a la distribución inicial fijada.

Estos pasos son suficientes para simular un MDP con *Simplex Features* y generar un dataset que pueda ser usado por los algoritmos para offline learning o para planear en el ambiente en online learning.

4.2 Evaluación de los Algoritmos

Una forma sencilla obtener una métrica de qué tan bien funciona un algoritmo aproximando el MDP real, es ver qué tan bien se aproximan cada una de las matrices de transición inducidas por cada acción. En nuestro MDP simulado con la metodología anterior, para cada a su matriz inducida será $M_a := M[:, :, a]^T \cdot U$. Llamemos $\overline{M}_a$, $\hat{M}_a$ y $\tilde{M}_a$ a las aproximaciones dadas por el algoritmo 4, FLAMBE y REP-UCB respectivamente. Entonces, las métricas de error a considerar serán $\overline{\varepsilon} = \sum_{a\in\mathcal{A}} \parallel M_a - \overline{M}_a \parallel_{FB}$, $\hat{\varepsilon} = \sum_{a\in\mathcal{A}} \parallel M_a - \hat{M}_a \parallel_{FB}$ y $\tilde{\varepsilon} = \sum_{a\in\mathcal{A}} \parallel M_a - \tilde{M}_a \parallel_{FB}$. En general ε_{FB} será de las aproximaciones con respecto a la norma de Frobenius.

4.2.1 Evaluación con Diferentes Políticas Exploratorias

Lo primero era ver qué tan bien se comportaban los MLE de FLAMBE y REP-UCB aproximando nuestro MDP si solo se tenía una base de datos obtenida siguiendo una política exploratoria uniforme. Esto es relevante, ya que el algoritmo 4 basado en sGS-ADMM es enteramente diseñado para ser offline, por ende, esta es una primera

comparación justa entre la aproximación por clases de funciones y la hecha por sGS-ADMM. Cabe recordar que se usaron las mismas clases de funciones (redes neuronales) tanto para FLAMBE como para REP-UCB, por lo que los resultados aquí expuestos son válidos para ambas.

Para un ambiente con 100 estados, 10 acciones y rango 3 se generó un dataset con $30 \cdot \lceil 10 \cdot 3 \cdot 100 \log(100) \rceil$ muestras.

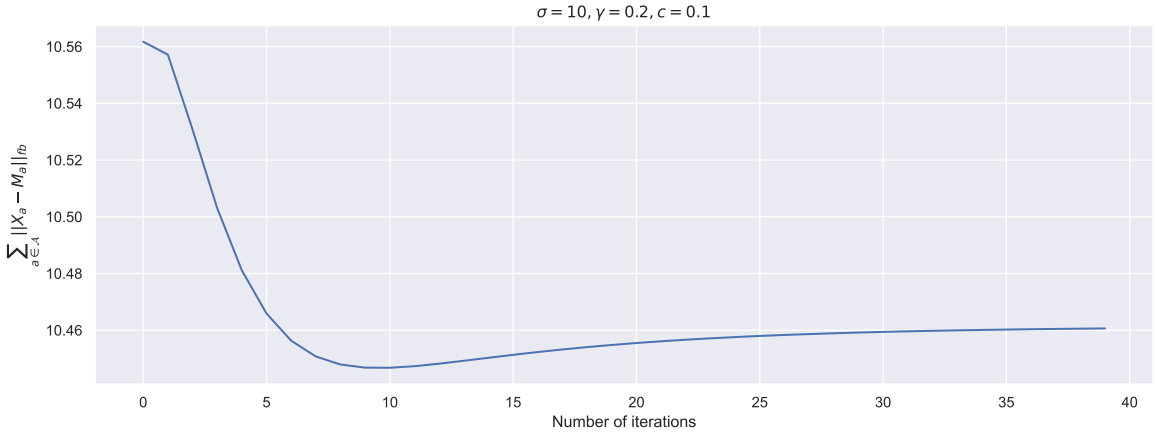


Figura 4.1: Evolución de ε_{FB} al correr el algoritmo 4 sobre el dataset de entrenamiento.

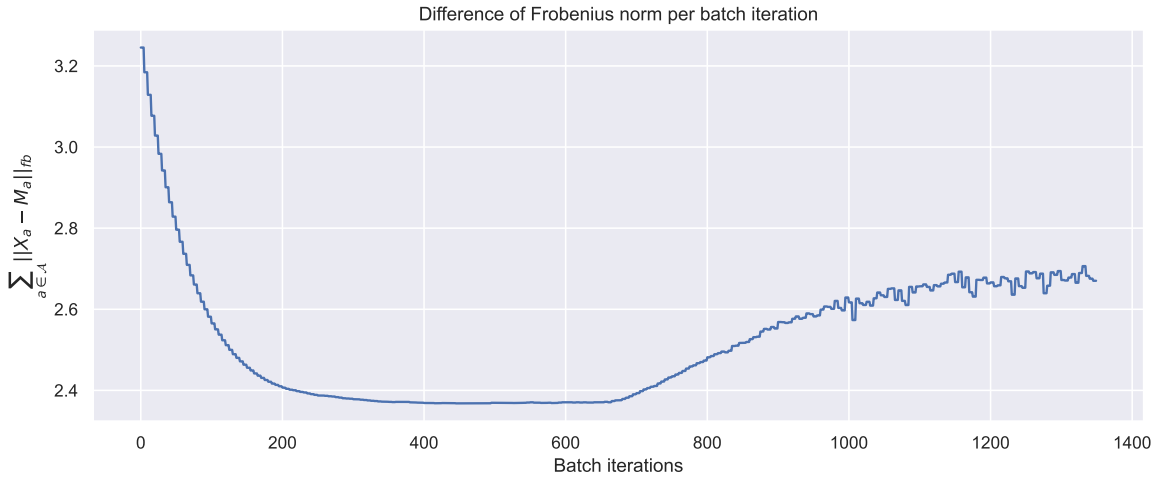


Figura 4.2: Evolución de ε_{FB} al usar redes neuronales en el dataset de entrenamiento para aproximar ϕ y μ.

Note que si bien el error en la figura 4.1 es bastante estable y converge luego de pocas iteraciones del algoritmo, este llega a un valor más alto que aquel alcanzado por las redes neuronales. De hecho, casi que desde las primeras iteraciones, las redes neuronales obtienen mejores aproximaciones en términos de ε_{FB} que usar el algoritmo 4. Esto se podría deber a que las arquitectura de las redes neuronales implementadas capturan una información importante que sGS-ADMM no puede capturar, y es el hecho que cada matriz inducida M_a se puede factorizar como la multiplicación de dos matrices M_{a_1} y M_{a_2}, donde M_{a_2} es idéntica para todas las acciones (en nuestro MDP

simulado esta matriz corresponde a U, la matriz inducida por la función μ). Cabe destacar que en la figura 4.2, la capa de embedding tenía exactamente 3 neuronas, las mismas que el rango del MDP real, en la práctica este parámetro no es conocido, solo se supone que es mucho más pequeño que el número de estados. Para comprobar la efectividad de las redes neuronales con un tamaño de embedding más grande al real, se volvió a correr el experimento con una capa de embedding de tamaño 15, que es 5 veces mayor al rango real del MDP. En la figura 4.3 se puede ver como, incluso suponiendo un rango mucho mayor al real, las aproximaciones obtenidas siguen siendo buenas.

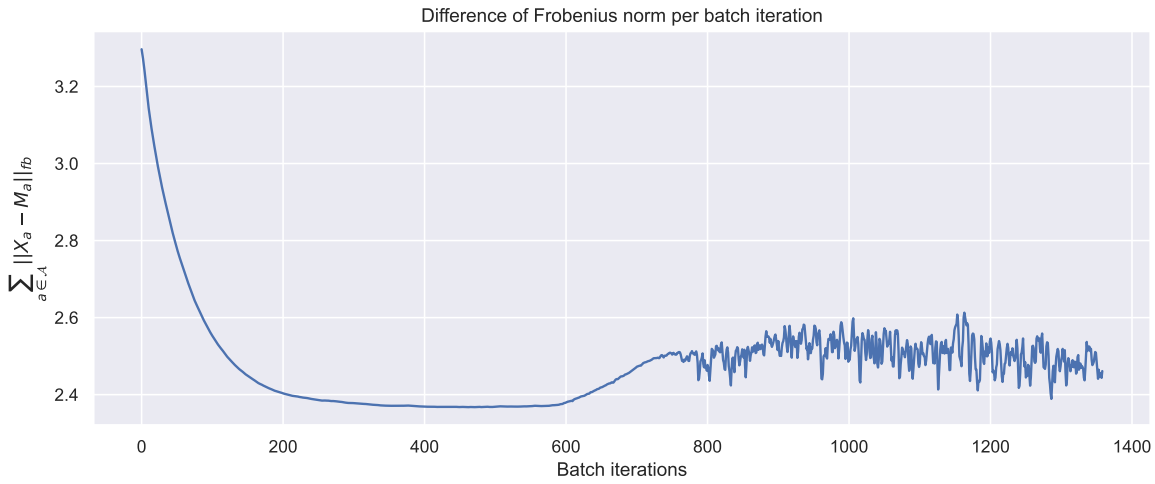


Figura 4.3: Evolución de ε_{FB} al usar redes neuronales en el dataset de entrenamiento para aproximar ϕ y μ. En este caso la capa de embbeding es 5 veces más grande que el rango real del MDP.

En general se puede apreciar en estos resultados que las redes neuronales hacen un trabajo mucho mejor aproximando nuestro MDP que el enfoque derivado de usar sGS-ADMM múltiples veces. Por otro lado, se puede pensar en este experimento como una implementación de FLAMBE que usa una política exploratoria uniforme.

Una observación importante con respecto a la figura 4.2 y 4.3 es que es evidente como el algoritmo a medida que avanza se vuelve inestable con respecto a su comportamiento para ε_{FB}. Esto puede deberse a que a medida que se obtienen más muestras uniformes, la red intenta modelar el sistema con muestras que no son representativas de la dinámica de bajo rango, por ende, si bien logra que la función de pérdida baje, esto es a costa del desempeño de ε_{FB}, la idea entonces de FLAMBE y REP-UCB es que al explorar usando los algoritmos de planeación, las muestras obtenidas serán representativas de la dinámica de bajo rango del sistema. En este caso usar los planners anteriormente mencionados implica que ε_{FB} se estabiliza por lo que añadir más datos mejora el modelo a diferencia del caso de solo usar una política exploratoria uniforme, este hecho es evidente en la figura 4.4.

Por otro lado, también se esperaría que REP-UCB tenga un comportamiento similar ya que la filosofía exploratoria de este algoritmo, en el fondo, es muy parecida al Elliptical Planner de FLAMBE. Y, de hecho, se comporta de forma no muy lejana a FLAMBE. Recordemos que la forma en la que REP-UCB planea en el ambiente para cada iteración es resolviendo el problema de encontrar una política optimal sobre el

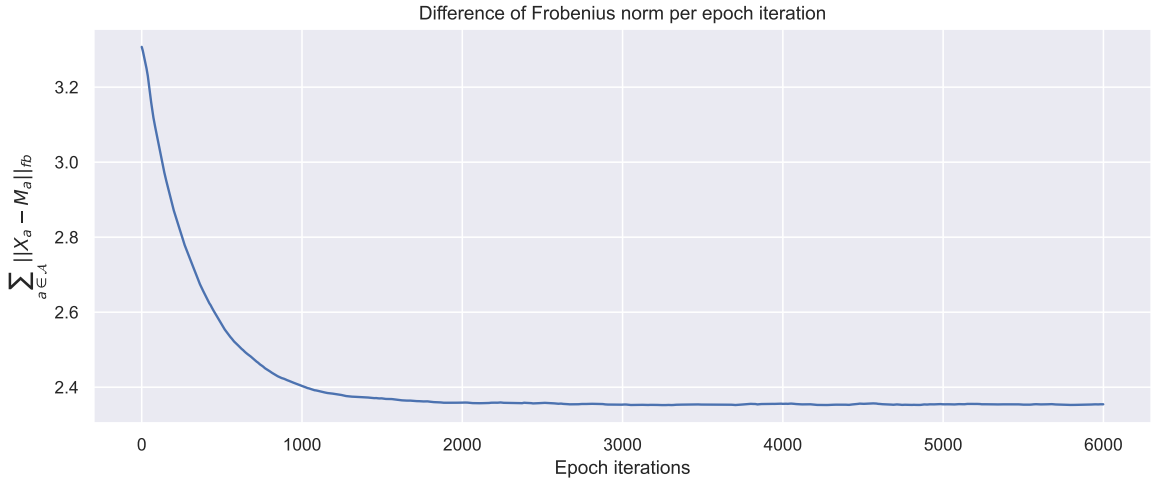


Figura 4.4: Evolución de ε_{FB} al usar redes neuronales en el dataset de entrenamiento para aproximar ϕ y μ y, además, usar el Simplex Planner como política exploratoria

MDP modelado por la red neuronal y agregando un bono de exploración a la señal original de recompensa. Este proceso se puede hacer implementando algo como Value Iteration o Policy Iteration y, en general, ambas aproximaciones requieren un número muy grande de iteraciones para poder converger a una política óptima, aunque en el caso particular de nuestro MDP simulado, la convergencia fue muy rápida usando Policy Iteration. Los tiempos de ejecución por cada iteración de los algoritmos están consignados en el cuadro 1.

Algoritmo	**Segundos/ Iteración**	**Paralelizable**	**Iteraciones**
sGS-ADMM	35	Sí	~ 30
FLAMBE	0.097	Sí	~ 2000
REP-UCB	0.3	No	~ 2000

Cuadro 1: Comparativa de tiempos de ejecución de los algoritmos para un *Simplex Feature* MDP con 100 estados, 10 acciones y rango 3.

Entonces, en la figura 4.5 se muestran el comportamiento de ε_{FB} al ejecutar REP-UCB en un número de muestras equivalente al usando en la figura 4.4.

Vemos que, de hecho, se comporta de forma similar a como se comporta FLAMBE. Por último, recordemos que REP-UCB era el único algoritmo que daba cotas estadísticas en cuanto a la función de valor obtenida luego de correr el algoritmo. En el cuadro 2 se muestra la diferencia promedio entre la función de valor óptima dada por los algoritmos y la real calculada con el MDP original. Como referencia, el valor máximo que alcanza la *Value Function* real era de aproximadamente 10 (La recompensa era una función logarítmica sobre el número de cada estado y acción para evitar que explotaran los valores si $\mathcal{X}$ y $\mathcal{A}$ eran muy grandes).

Estas funciones de valor en el caso de sGS-ADMM y FLAMBE se obtienen resolviendo un problema lineal de 100 ecuaciones y 100 variables (tantas como estados hay) para cada política hasta que haya convergencia. Cabe mencionar que el condicionamiento de este problema se notó dependía fuertemente del valor de descuento que se

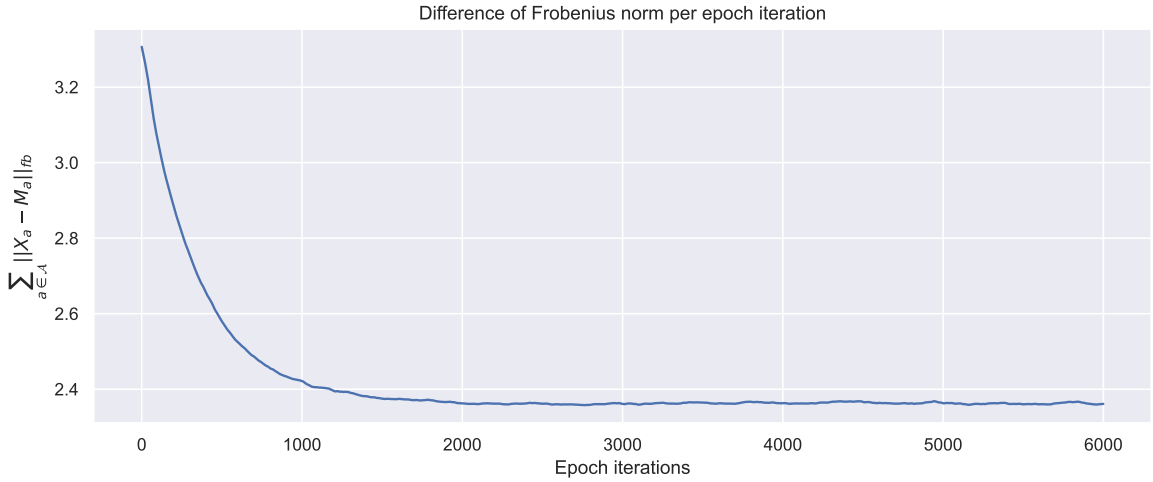


Figura 4.5: Evolución de ε_{FB} al usar redes neuronales en el dataset de entrenamiento para aproximar ϕ y μ y, además, usar el planeador de REP-UCB.

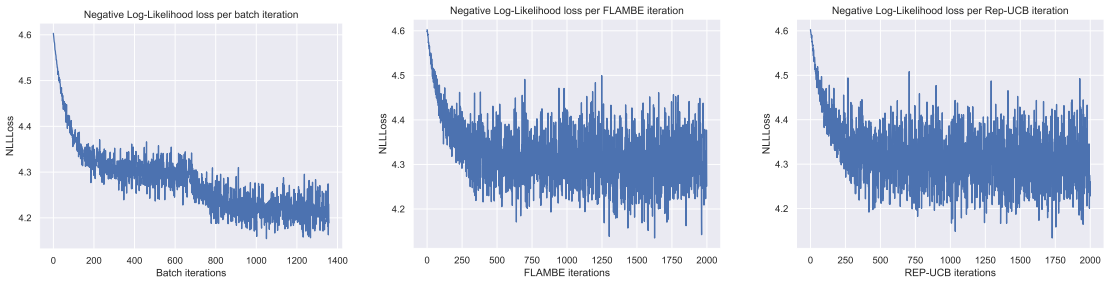
Algoritmo	Mean Diff Value Func
sGS-ADMM	0.0431
NN Exploración Uniforme	0.0347
FLAMBE	0.0338
REP-UCB	0.0339

Cuadro 2: Comparativa de la diferencia promedio entre las *Value Functions* calculadas con los algoritmos y la calculada con el MDP real.

usara, ya que si este número era 1, el número de condicionamiento de la matriz que representaba el sistema de ecuaciones era muy alto (en el orden de 10^{20}) volviéndolo intratable en la práctica. Este número, afortunadamente, decrecía exponencialmente mientras el valor de descuento se alejara de 1, tan así que si dicho descuento era 0,9 el condicionamiento ya era cercano a 30, que lo volvía mucho más tratable. En los experimentos de este trabajo se usó un descuento de 0,5, el cuál tenía un condicionamiento muy bajo, por ende, los resultados eran bastante exactos.

Como último resultado, no está de más ver el comportamiento de la función de costo de la red neuronal con cada uno de los algoritmos, esto se aprecia en la figura 4.6.

Se puede observar que la política exploratoria uniforme hace que la función de costo sea mucho menos inestable, esto se puede deber a que los algoritmos de planeación de FLAMBE y REP-UCB fuerzan al modelo a observar direcciones del espacio latente que no había visto antes, por ende, la red está aprendiendo de muestras fundamentalmente distintas a las que ha visto con anterioridad y, en estos momentos, la función de costo dará un salto ya que como la red no ha visto nada similar antes, no puede hacer buenas predicciones en ese instante, pero es justamente esta particularidad la que hace que luego ε_{FB} se comporte mucho mejor en FLAMBE y REP-UCB que solo con una política exploratoria uniforme.



(a) Exploración Uniforme (b) FLAMBE (c) REP-UCB

Figura 4.6: Comportamiento de la función de costo en nuestra red neuronal para los distintos algoritmos

El código con la implementación de todos los algoritmos y simulaciones aquí expuestas está en un repositorio de código abierto que el lector puede visitar en el siguiente link.

5 Conclusiones

Podemos concluir que los enfoques basados en modelar la estructura misma de los MDPs de bajo rango son mucho más efectivos en aproximar dicha estructura que otros más generales como el algoritmo 4 derivado de la implementación tradicional de sGS-ADMM.

Por otro lado, si bien REP-UCB tiene cotas teóricas un poco mejores que las de FLAMBE, en la práctica la escogencia de las clases de funciones Φ y Υ es mucho más importante. Además, la implementación de REP-UCB se vuelve más compleja que la de FLAMBE por el hecho de depender de un método como Policy Iteration para determinar su política exploratoria, esto debido a que encontrar una política optimal puede ser algo que si bien puede converger rápido en algunos casos, también está expuesto a una lenta convergencia y la diferencia entre una y otra situación no es muy evidente, por ende, se tiene que imponer un número máximo de iteraciones si no se quiere depender del azar en este algoritmo. Incluso al imponer dicha cota máxima de iteraciones, se notó que FLAMBE seguía siendo más rápido que REP-UCB.

Además, las técnicas tratadas en este trabajo serán útiles en situaciones para las cuales se cuenta solo con un acceso limitado al MDP original, por lo que resulta óptimo modelar el MDP antes de intentar buscar políticas óptimas sobre él. Luego de tener el modelo del ambiente obtenido con un número reducido de muestras, se pueden probar tantos algoritmos como se quiera para resolver el problema de RL.

En general los resultados experimentales obtenidos concuerdan con los descritos por la teoría y se logró estudiar y entender aquellos detalles y minucia teórica y práctica de cada uno de los algoritmos en distintas situaciones.

6 Agradecimientos

Agradezco a todos los amigos y familiares que contribuyeron positivamente a lo largo de mi proceso universitario, en especial a mis padres: Remberto y Patricia; mi hermana Luisa y su esposo Carlos Mario; mi hermano, Benrique; mis primos: Mary, Carly, Carlos Alberto, Maicol y Andrés; mis tías: Martha, Nivia, Rochy, Jennys, Yasmina y Rocío; tío Lucho y a mi querida abuela Rosario, todos quienes estuvieron a lo largo de mi proceso académico dándome constante apoyo de una u otra forma y motivándome a seguir dando lo mejor de mí ¡Gracias!. A mi asesor Mauricio Junca por su pertinente y constante guía le doy también las gracias.

7 Apéndice

7.1 Apéndice B: Resultados Adicionales sobre los Papers

7.1.1 B1: Análisis sGS-ADMM

Para calcular las iteraciones de y en el algoritmo 3 basta considerar la función

$$T(y) = \frac{\sigma}{2} \parallel A^*(y) + K \parallel^2 - \langle b, y \rangle,$$

donde $K = \Xi + S + \frac{X}{\sigma}$. Considere la función

$$\begin{aligned} f(x) &= \langle Ax + c, Ax + c \rangle \\ &= \langle Ax, Ax \rangle + 2\langle Ax, c \rangle + \parallel c \parallel^2 . \end{aligned}$$

Note que

$$(x+h)^t A^t A(x+h) = x^t A^t Ax + 2(A^t Ax)^t h + \epsilon(h)$$

donde $\epsilon(h) = h^t A^t Ah$ y como $\langle Ax, c \rangle = (Ac)^t x$, se sigue que:

$$\nabla f(x) = 2(A^t Ax + A^t c)$$

Por lo anterior, tenemos que el gradiente de T es:

$$\nabla T(y) = \sigma(\mathcal{A}\mathcal{A}^* y + \mathcal{A}K) - b.$$

Para obtener el óptimo, igualamos este gradiente a 0. Con esto el óptimo es $y = (\mathcal{A}\mathcal{A}^*)^{-1}(\frac{b}{\sigma} - \mathcal{A}K)$. En conclusión

$$\begin{aligned} y^{k+1} &= \frac{1}{\sigma}(\mathcal{A}\mathcal{A}^*)^{-1}(b - \mathcal{A}(X^k - \sigma(\Xi^{k+1} + S^k))), \\ y^{k+\frac{1}{2}} &= \frac{1}{\sigma}(\mathcal{A}\mathcal{A}^*)^{-1}(b - \mathcal{A}(X^k - \sigma(\Xi^k + S^k))) \end{aligned}$$

Además se puede probar que $\mathcal{A}\mathcal{A}^* y = py$.
Las iteraciones de Ξ se calculan resolviendo este problema de optimización:

$$\min_{\Xi}\{g^*(-\Xi) + \frac{\sigma}{2} \parallel \Xi + R^k \parallel^2\},$$

donde $R^k = \mathcal{A}^*(y^{k+\frac{1}{2}} + S^k + X^k/\sigma)$

Teorema 7.1. (Identidad de Moreau, Teorema 31.5 [Rockafellar, 1970]) Sea f un función propia convexa cerrada sobre $\mathbb{R}^n$. Entonces

$$\inf_x\{f(x) + \frac{1}{2} \parallel z - x \parallel^2\} + \inf_{x^*}\{f^*(x^*) + \frac{1}{2} \parallel z - x^* \parallel\} = \frac{1}{2} \parallel z \parallel^2$$

donde ambos ínfimos son finitos, únicos y se alcanzan. Los únicos vectores x y x^* en los cuales se alcanzan los ínfimos respectivamente para un z dado son lo únicos tales que

$$x + x^* = z \quad x^* \in \partial f(x)$$

y están dados por

$$x = \nabla \operatorname{ínf}_x \{f(x) + \frac{1}{2} \parallel z - x \parallel^2\}$$
$$x^* = \nabla \operatorname{ínf}_{x^*} \{f^*(x^*) + \frac{1}{2} \parallel z - x^* \parallel\}.$$

Utilizando la identidad de Moreau tenemos que

$$\Xi^{k+1} = \frac{1}{\sigma}(Z^k - \sigma R^k)$$

donde

$$Z^k = \operatorname{mín}_Z \{\sigma g(Z) + \frac{1}{2} \parallel Z - \sigma R^k \parallel^2\}$$

para el cual se puede demostrar que

$$Z_{ij}^k = \begin{cases} \frac{\sigma R_{ij}^k + \sigma\sqrt{(R_{ij}^k + 4n_{ij}/(n\sigma))^2}}{2} & (i,j) \in \Omega \\ \sigma \operatorname{máx}(R_{ij}^k, 0) & (i,j) \notin \Omega \end{cases}$$

Las iteraciones de S se pueden calcular resolviendo el problema de optimización

$$\arg\operatorname{mín}\{\frac{\sigma}{2} \parallel S - W^k \parallel : \parallel S \parallel_2 \leq c\}$$

donde $W^k = -(\Xi^{k+1} + \mathcal{A}^*(y^{k+1}) + X^k/\sigma)$. (Lema 2.1 [Jiang et al., 2014]). Sea $Y \in \mathbb{R}^{n\times m}$ y $Y = U\Sigma_Y V^t$ su SVD. Entonces la única solución del problema

$$\arg\operatorname{mín}\{\parallel X - Y \parallel^2 : \parallel X \parallel_2 \leq \rho\}$$

es $\hat{X} = U \operatorname{mín}(\Sigma_Y, \rho)V^t$, donde $\operatorname{mín}(\Sigma_Y, \rho) = \operatorname{Diag}(\operatorname{mín}\{\sigma_1, \rho\}, ..., \operatorname{mín}\{\sigma_p, \rho\})$.

Demostración. Claramente si el problema tiene una solución esta será única. Defina

$$A_1 = \{i : \sigma_i(Y) > \rho\} \text{ y } A_2 = \{i : \sigma_i(y) \leq \rho\}$$

Sea Z un valor factible con SVD $Z = U_1\Sigma_Z V_1^*$, por suposición tenemos que $\sigma_i(Z) \leq \rho$ para $1 \leq i \leq p$. Entonces por (Ejercicio IV.3.5, [Bhatia, 1997]) como $\parallel \cdot \parallel$ es unitariamente invariante tenemos

$$\begin{aligned} \parallel Z - Y \parallel^2 &\geq \parallel \Sigma_Z - \Sigma_Y \parallel \\ &\geq \sum_{i\in A_1} (\sigma_i(Y) - \sigma_i(Z)) + \sum_{i\in A_2} (\sigma_i(Y) - \Sigma_i(Z)) \\ &\geq \sum_{i\in A_1} (\sigma_i(Y) - \rho) = \parallel Y - \hat{X} \parallel^2 \end{aligned}$$

□

Por el Lema 7.1.1 tenemos que la expresión explícita para S^{k+1} es:

$$S^{k+1} = U_k \operatorname{mín}(\Sigma_k, c)V_k^t$$

donde $W_k = U_k \Sigma_k V_k^t$ y $\text{mín}(\Sigma_k, c) = \text{Diag}(\text{mín}(\alpha_1, c), ..., \text{mín}(\alpha_n, c))$ con $\alpha_1 \geq ... \geq \alpha_n$ son los valores singulares de W_k.

En general la solución del problema dual sólo da un cota inferior al problema de optimización. Sin embargo, en este caso las parejas de soluciones primal-dual asociadas a los problemas (1) y (3) satisfaces el sistema KKT.

$$0 \in R(X, \Xi, S), \quad \mathcal{A}(X) = b, \quad \Xi + \mathcal{A}^*(y) + S = 0$$

con

$$R(X, \Xi, S) = \begin{bmatrix} \Xi + \partial g(X) \\ X + \partial \delta(\| S \|_2 \leq c) \end{bmatrix}$$
$$(X, \Xi, S) \in \text{dom}(g) \times \mathbb{R}^{p \times p} \times \{S \in \mathbb{R}^{p \times p} : \| S \|_2 \leq c\}$$

Además el siguiente resultado garantiza que el Algoritmo aproxima una solución tanto del problema dual como del problema primal.

Teorema 7.2. (Teorema EC.3 [Zhu et al., 2021]) Suponga que el conjunto de soluciones de (1) y (3) son no vacíos. Sea $\{(\Xi^k, y^k, S^k, X^k)\}_{k \in \omega}$ la sucesión generada en el Algoritmo. Si $\gamma \in (0, (1 + \sqrt{5})/2)$ entonces la sucesión $\{(\Xi^k, y^k, S^k)\}_{k \in \omega}$ converge a una solución óptima de (3) y $\{X^k\}_{k \in \omega}$ converge a una solución óptima de (1).

7.1.2 B2: FLAMBE

Teorema 7.3 (Fórmula de Sherman-Morrison)**.** Dados vectores $v_j \in \mathbb{R}^d, 0 < j \leq k$, I la identidad $d \times d$ y $\alpha > 0$. Definimos $S_k := \alpha I + \sum_{i=1}^k v_i v_i^T$. Entonces, se puede encontrar iterativamente la inversa de S_k de la siguiente forma:

$$S_k^{-1} = S_{k-1}^{-1} - \frac{w_k w_k^T}{1 + v_k^T w_k}, w_k := S_{k-1}^{-1} v_k \tag{17}$$

Demostración. Sea $X := A + uv^t$, para vectores $v, u \in \mathbb{R}^n$ y una matriz invertible $A \in \mathbb{R}^{n \times n}$ y sea Y el resultado de aplicar la fómula de Sherman-Morrison a X, veamos entonces que $XY = I$.

$$\begin{aligned}
XY &= \left(A + uv^\intercal\right)\left(A^{-1} - \frac{A^{-1}uv^\intercal A^{-1}}{1 + v^\intercal A^{-1}u}\right) \\
&= AA^{-1} + uv^\intercal A^{-1} - \frac{AA^{-1}uv^\intercal A^{-1} + uv^\intercal A^{-1}uv^\intercal A^{-1}}{1 + v^\intercal A^{-1}u} \\
&= I + uv^\intercal A^{-1} - \frac{uv^\intercal A^{-1} + uv^\intercal A^{-1}uv^\intercal A^{-1}}{1 + v^\intercal A^{-1}u} \\
&= I + uv^\intercal A^{-1} - \frac{u\left(1 + v^\intercal A^{-1}u\right)v^\intercal A^{-1}}{1 + v^\intercal A^{-1}u} \\
&= I + uv^\intercal A^{-1} - uv^\intercal A^{-1} \\
&= I
\end{aligned}$$

□

7.1.3 B3: REP-UCB

El algoritmo 9 colecta un número de muestras n en el orden de

$$O\bigg(\frac{d^4|\mathcal{A}|^2\log\left(|\mathcal{M}|/\delta\right)}{(1-\gamma)^5\varepsilon^2}\cdot\nu\bigg),$$

donde ν está dada por:

$$\nu := O\bigg(\ln\Big(\frac{d^4|\mathcal{A}|^2\log\left(|\mathcal{M}|/\delta\right)}{(1-\gamma)^5\gamma\varepsilon^2}\ln^2\Big(1+\frac{d^4|\mathcal{A}|^2\log\left(|\mathcal{M}|/\delta\right)}{(1-\gamma)^5\varepsilon^2}\Big)\Big)\cdot\ln^2\Big(1+\frac{d^4|\mathcal{A}|^2\log\left(|\mathcal{M}|/\delta\right)}{(1-\gamma)^5\varepsilon^2}\Big)\bigg),$$

Referencias

[Agarwal et al., 2020] Agarwal, A., Kakade, S., Krishnamurthy, A., and Sun, W. (2020). Flambe: Structural complexity and representation learning of low rank mdps.

[Albus, 1981] Albus, J. S. (1981). Brains, behavior, and robotics.

[Amani et al., 2022] Amani, S., Yang, L. F., and Cheng, C.-A. (2022). Provably efficient lifelong reinforcement learning with linear function approximation. *arXiv preprint arXiv:2206.00270*.

[Berner et al., 2021] Berner, J., Grohs, P., Kutyniok, G., and Petersen, P. (2021). The modern mathematics of deep learning.

[Bhatia, 1997] Bhatia, R. (1997). *Matrix analysis*, volume 169 of *Graduate Texts in Mathematics*. Springer-Verlag, New York.

[Boţ et al., 2009] Boţ, R. I., Grad, S.-M., and Wanka, G. (2009). *Duality in vector optimization*. Vector Optimization. Springer-Verlag, Berlin.

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press, Cambridge.

[Chen et al., 2016] Chen, C., He, B., Ye, Y., and Yuan, X. (2016). The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent. *Math. Program.*, 155(1-2, Ser. A):57–79.

[Deng and Huang, 2012] Deng, K. and Huang, D. (2012). Model reduction of markov chains via low-rank approximation. In *2012 American Control Conference (ACC)*, pages 2651–2656. IEEE.

[Du et al., 2019] Du, S. S., Krishnamurthy, A., Jiang, N., Agarwal, A., Dudík, M., and Langford, J. (2019). Provably efficient rl with rich observations via latent state decoding.

[Fawzi et al., 2022] Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatain, M., Novikov, A., R Ruiz, F. J., Schrittwieser, J., Swirszcz, G., et al. (2022). Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53.

[Fortin and Glowinski, 1983] Fortin, M. and Glowinski, R. (1983). *Augmented Lagrangian methods*, volume 15 of *Studies in Mathematics and its Applications*. North-Holland Publishing Co., Amsterdam. Applications to the numerical solution of boundary value problems, Translated from the French by B. Hunt and D. C. Spicer.

[Glowinski, 2008] Glowinski, R. (2008). *Numerical methods for nonlinear variational problems*. Scientific Computation. Springer-Verlag, Berlin. Reprint of the 1984 original.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Hestenes, 1969] Hestenes, M. R. (1969). Multiplier and gradient methods. *J. Optim. Theory Appl.*, 4:303–320.

[Jiang et al., 2014] Jiang, K., Sun, D., and Toh, K.-C. (2014). A partial proximal point algorithm for nuclear norm regularized matrix least squares problems. *Math. Program. Comput.*, 6(3):281–325.

[Li et al., 2016] Li, X., Sun, D., and Toh, K.-C. (2016). A Schur complement based semi-proximal ADMM for convex quadratic conic programming and extensions. *Math. Program.*, 155(1-2, Ser. A):333–373.

[Lin, 2016] Lin, Z. (2016). A review on low-rank models in data analysis. *Big Data & Information Analytics*, 1(2&3):139.

[Luenberger, 1969] Luenberger, D. G. (1969). *Optimization by vector space methods*. John Wiley & Sons, Inc., New York-London-Sydney.

[Misra et al., 2019] Misra, D., Henaff, M., Krishnamurthy, A., and Langford, J. (2019). Kinematic state abstraction and provably efficient rich-observation reinforcement learning.

[Montague, 1999] Montague, P. R. (1999). Reinforcement learning: an introduction, by sutton, rs and barto, ag. *Trends in cognitive sciences*, 3(9):360.

[Norris, 1998] Norris, J. R. (1998). *Markov chains*, volume 2 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, Cambridge. Reprint of 1997 original.

[Powell, 1969] Powell, M. J. D. (1969). A method for nonlinear constraints in minimization problems. In *Optimization (Sympos., Univ. Keele, Keele, 1968)*, pages 283–298. Academic Press, London.

[Rendle et al., 2010] Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. (2010). Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 811–820, New York, NY, USA. Association for Computing Machinery.

[Rockafellar, 1970] Rockafellar, R. T. (1970). *Convex analysis*. Princeton Mathematical Series, No. 28. Princeton University Press, Princeton, N.J.

[Sherman and Morrison, 1950] Sherman, J. and Morrison, W. (1950). Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, 21(1):124–127.

[Sherstov and Stone, 2005] Sherstov, A. A. and Stone, P. (2005). Function approximation via tile coding: Automating parameter choice. In Zucker, J.-D. and Saitta, L., editors, *Abstraction, Reformulation and Approximation*, pages 194–205, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Stiennon et al., 2020] Stiennon, N., Ouyang, L., Wu, J., Ziegler, D., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. F. (2020). Learning to summarize with human feedback. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc.

[Sutton, 1995] Sutton, R. S. (1995). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky, D., Mozer, M., and Hasselmo, M., editors, *Advances in Neural Information Processing Systems*, volume 8. MIT Press.

[Udell and Townsend, 2019] Udell, M. and Townsend, A. (2019). Why are big data matrices approximately low rank? *SIAM Journal on Mathematics of Data Science*, 1(1):144–160.

[Uehara et al., 2021] Uehara, M., Zhang, X., and Sun, W. (2021). Representation learning for online and offline rl in low-rank mdps.

[Whiteson et al., 2007] Whiteson, S., Taylor, M. E., and Stone, P. (2007). Adaptive tile coding for value function approximation. Technical Report AI-TR-07-339, University of Texas at Austin.

[Zhu et al., 2021] Zhu, Z., Li, X., Wang, M., and Zhang, A. (2021). Learning markov models via low-rank optimization. *Operations Research*.