Instituto Tecnológico Autónomo de México

Diseño de funciones de recompensas para Q-Learning usando Modelos Extensos de Lenguaje

Tesis

que para obtener el título de

Licenciado en Ciencia de Datos

Presenta

Javier Nieto Merodio

Asesora

Mtro. Mario Vázquez Corte

Ciudad de México 2025

≪Con fundamento en los artículos 21 y 27 de la Ley Federal del Derecho de Autor y como titular de los derechos moral y patrimonial de la obra titulada "**Diseño de funciones de recompensas para Q-Learning usando Modelos Extensos de Lenguaje**", otorgo de manera gratuita y permanente al Instituto Tecnológico Autónomo de México y a la Biblioteca Raúl Bailléres Jr., la autorización para que fijen la obra en cualquier medio, incluido el electrónico, y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por tal divulgación una contraprestación.≫

Fecha

Javier Nieto Merodio

A mi abuela,
por enseñarme que aprender es divertido.

Agradecimientos

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Resumen

texto.

Summary

text.

Índice general

Índice de cuadros

Índice de figuras

Introducción

Cinco páginas de extensión.

- Planteamiento del problema
- ¿Por qué es importante? ¿Qué limitaciones tienen los enfoques actuales?
- ¿Qué propones hacer diferente?
- Hipótesis y objetivo general
- Breve descripción de tu metodología
- Estructura del documento

Capítulo 1

Marco Teórico

Sección para explicar los detalles técnicos y origen detrás del uso de modelos extensos de lenguaje para diseñar funciones de recompensa.

1.1. Aprendizaje por refuerzo

1.1.1. Definición y origen

El aprendizaje por refuerzo, uno de los paradigmas principales del aprendizaje de máquina, puede ser definido de distintas formas. Sutton & Barto lo definían en su biblia de la materia, *Reinforcement Learning: An Introduction* como aprender qué hacer y asignar acciones a escenarios para maximizar una señal de recompensas [12]. De forma más técnica, Kaelbing et. al. lo definían como un conjunto de problemas en el que un agente debe aprender mediante la prueba y el error a interactuar con un ambiente dinámico [5].

Realmente, más allá de los elementos técnicos y el rigor del código, el aprendizaje por refuerzo puede ser identificado en su núcleo a cómo aprende el ser humano. Desde una experiencia tan temprana como aprender que las piezas de un rompecabezas solo encajaban en una posición luego de intentar forzar una pieza en otro lugar, hasta la lección universal del ITAM que si quieres pasar los exámenes de Cálculo de Probabilidades tienes que hacer constantemente ejercicios, tras un mal desempeño luego de hacer ejercicios unas noches antes. El ser humano recibe muchos de sus aprendizajes por medio de prueba y error, interactuando con distintos ambientes, ya sea un rompecabezas o un examen de estadística.

Habiendo mencionado lo anterior, el origen del aprendizaje por refuerzo puede ser atribuido a áreas de estudio fuera de la computación, y más del lado de la psicología. Específicamente, la psicología conductual tuvo aportes esfuerzos notables en el siglo XX con investigaciones del aprendizaje de animales mediante refuerzo. Entre sus principales referentes, Edward Thorndike destacó con su *ley del efecto* que una acción seguida de un resultado satisfactorio es

probable que vaya a repetirse. Luego, Burrhus Frederick Skinner continuó estas aportaciones, trabajando para dar forma a un aprendizaje mediante refuerzos sucesivos [12].

Luego, como nota Kaelbing en *Reinforcement Learning: A Survey*, estas aportaciones de la psicología fueron seguidas por las aportaciones de varios personajes a la investigación de la programación dinámica, como Richard Bellman y Ronald Howard [5]. Estos esfuerzos de la programación dinámica de la mano de la idea de la psicología conductista se ven conectados en el trabajo de Chris Watkins de 1989 [16], en la cual un agente aprende mediante recompensas en un ambiente Markoviano.

Para finales de los años 90, el aprendizaje por refuerzo ya era un campo propio, con diferentes libros y artículos, así como conferencias de investigación [12]. Hoy en día, vemos el aprendizaje por refuerzo ser usado en distintos escenarios, desde el entrenamiento de bots para videojuegos hasta la exploración de escenarios para teoría de juegos buscando un agente que desarrolle su propia política de acción. Si bien, no es un paradigma tan estudiado como el aprendizaje supervisado y no supervisado, es un campo de estudio con infinitas aplicaciones y un uso cada vez más elevado.

1.1.2. Elementos clave

Ahora bien, para poder entender mejor el aprendizaje por refuerzo y eventualmente *Q-Learning*, se necesitan aprender los elementos clave del campo para comprender la composición de estos programas. Para la definición y notación de los componentes se usaron. las notas de Sutton & Barton [12] y Szevespari [13].

Nuestro primer elemento es el estado, s el cual nos define en punto dado cómo es que se ve el ambiente con el que el agente está

interactuando. Al conjunto de estados en el que un agente puede encontrarse se le denota con el símbolo S. Habiendo definido dicho elemento, es pertinente explicar qué es un Proceso de Markov. Según Sutton & Barto, es aquel que satisface la propiedad de Markov. No muy explicativo. Ahora bien, la propiedad de Markov se cumple cuando el siguiente estado depende únicamente del estado actual y la acción ejecutada. Más formalmente, *el futuro es independiente del pasado dado el presente*. En términos de cálculo de probabilidades,

$$\mathbb{P}(S_{t+1} \mid S_t) = \mathbb{P}(S_{t+1} \mid S_1, S_2, \ldots, S_t)$$

. Un ejemplo donde se cumple la propiedad de Markov serían los juegos de ajedrez, ya que en cualquier punto que se tome un juego cualquiera de ajedrez, no se necesita más información que el estado actual con la posición de las piezas de ambos colores para ejecutar una siguiente acción. Por otro lado, si yo metiera a calentar una pizza a un horno tradicional, y mi estado fuera la posición y el color de la pizza, no estaría cumpliendo la propiedad de Markov, ya que necesitaría conocer de estados anteriores y el tiempo transcurrido para sacarla en el momento correcto.

El siguiente elemento a describir sería la acción a, siendo A el conjunto de todas las acciones posibles que puede tomar un agente. Igualmente, podemos utilizar la notación de Szevespari para aquellas acciones que el agente pueda tomar dado un estado s como $A(s)$.

Luego, tenemos la función de recompensas $R(s, a)$ la cual nos ofrece un escalar para demostrar el valor de haber realizado una acción a en un estado s. Es en esta función donde se localiza el núcleo del aprendizaje por refuerzo, y con la cual podemos darle forma al aprendizaje deseado, de la misma forma que Skinner intentaba dar forma al aprendizaje de los animales.

Luego, tenemos la función de transición $P(s' \mid s, a)$, la cual nos dice dado un estado y una acción tomada en ese estado, la probabilidad de pasar a un estado s'. En un entorno estocástico, tendríamos distintos estados posibles de alcanzar tomada dicha acción en el estado s, cada uno con cierta probabilidad.

Ahora, definimos la política del agente $\pi(a|s)$ como la asignación de probabilidades a tomar ciertas acciones dado cierto estado. En sí, la política es el núcleo de los aprendizajes de nuestro agente. Dependiendo en qué tanto explore o use su conocimiento (siguiente subsección), el agente decide qué acción tomar dado un estado.

Posteriormente, definimos el factor de descuento γ como el parámetro que quita valor a recompensas futuras y mantiene un aprendizaje enfocado al largo plazo, muy relacionado con la contribución inicial de Chris Watkins en 1989. Esta tasa, definida en $\gamma \in [0, 1]$, toma un valor 0 cuando solo valen las recompensas del hoy, y 1 el valorar por completo las recompensas de periodos futuros. Al representar el retorno total de las recompensas por etapa del aprendizaje $G_t = R_{t+1} + \gamma R_{t+2} + \cdots$ podemos entender mejor el funcionamiento de γ.

Finalmente, se introducen dos conceptos muy importantes: la función de valor y el valor-acción. La primera, definida de la siguiente forma:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

Nos da el valor esperado de seguir la política π estando en un estado s. Se puede apreciar que la tasa de descuento γ mide el valor de las recompensas futuras, siendo cada vez menor con el tiempo al aumentar su exponente y estando la tasa entre 0 y 1.

Por otro lado, la función valor-acción se define como:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k}R_{t+k+1}\,\middle|\,S_t = s, A_t = a\right]$$

De forma muy similar a la función valor, pretende hacer una valuación proyectada a futuro de las recompensas. La diferencia es que en esta segunda función no solo evalúa dado un estado, sino también al haber tomado una acción bajo la política. Esta nos sirve para evaluar qué acciones convendría tomar al agente dada una esperanza de recompensas acumuladas.

Cuadro 1.1. Elementos del aprendizaje por refuerzo

Notación	**Nombre**
s	Estado
S	Espacio de estados
a	Acción
A	Espacio de acciones
$A(s)$	Conjunto de acciones disponibles en s
$R(s,a)$	Función de recompensa
$P(s'\|s,a)$	Función de transición
$\pi(a\|s)$	Política
γ	Factor de descuento
G_t	Retorno total
$V^{\pi}(s)$	Función de valor
$Q^{\pi}(s,a)$	Función valor-acción

1.1.3. Explotación y exploración

Una vez entendido qué es el aprendizaje por refuerzo y sus elementos principales, surge una cuestión crítica que definirá el aprendizaje. ¿Qué tanto quiero explorar acciones, y qué tanto quiero aprovecharme de lo ya aprendido?

Por ejemplo, jugando un partido de futbol puedo decidir si patear la pelota de formas que conozco, o probar una nueva forma. Conforme vaya pateando la bola, puedo decidir abusar del conocimiento que ya tengo de una manera de golpear la pelota, o de aventurarme a conocer una nueva. Dependiendo qué tanto elija una u otra, mi aprendizaje tomará un camino u otro.

En estudios de aprendizaje por refuerzo, el problema que generalmente se usa para explicar el dilema es el *k-armed bandit*, detallado por Lattimore y Szevespari en *Bandit Algorithms* [7]. Este consiste en el siguiente escenario. Imaginemos que estamos en un establecimiento con k máquinas de traga-monedas que, al jalar la palanca de cada una, regresan una recompensa de acuerdo a una distribución de probabilidad fija y desconocida. Ahora, en cada paso del aprendizaje, podemos elegir de un conjunto de k acciones. Es decir, elegir qué traga-monedas usar. Eventualmente, nosotros queremos tener una política de acciones que maximice nuestra acumulación de recompensas. Para esto, tenemos que explorar distintas palancas. Supongamos que encontramos una máquina con un retorno de recompensas relativamente buenas. Podemos abusar del conocimiento de esta máquina, o explorar otras máquinas con vista a tener una mejor selección de máquinas y mayores recompensas, con el riesgo de agotar pasos explorando traga-monedas deficientes. De esta misma forma, tendremos que indicar a nuestros algoritmos qué tanto

queremos que en un ambiente explore nuevas acciones o explote su política para poder tener buenas recompensas en el futuro.

1.2. Q-Learning

1.2.1. Definición y origen

El aprendizaje por refuerzo, como paradigma del aprendizaje de máquina, abarca una amplia variedad de algoritmos. Para esta tesis, nos enfocamos particularmente en uno de ellos: *Q-Learning*. Introducido por primera vez por Chris Watkins en 1989, se definió en su momento como un algoritmo que permite al agente aprender a actuar de forma óptima en estados markovianos [16].

Años más tarde, Watkins junto con Dayan definían que este tipo de algoritmos implican que el agente emprenda su aprendizaje con la particularidad de no necesitar que el agente cree un mapa o modelo para el ambiente en el que se desarrolla. Recordando la notación antes definida, Watkins y Dayan buscaban que el agente aprendiera el valor de las parejas de acción y estado $Q(s, a)$ para llegar al máximo valor a largo plazo de recompensas acumuladas [15]. Desde 1989, Watkins, planteando a su vez las bases del aprendizaje por refuerzo en sí, buscaba con este algoritmo en particular una forma en la que los agentes pudieran aprender mediante recompensas diluidas en el tiempo. Más tarde, en nuestros experimentos, veremos cómo esta idea da forma a la base de la tesis central. De igual forma, nos ayudará a entender cómo poco a poco recompensas localmente bajas llevan a un agente a aprender al largo plazo a resolver una tarea aún más grande.

1.2.2. Descripción del algoritmo

Una vez explicada la definición del algoritmo, y entendiendo su propósito principal, ahora detallamos con pseudocódigo cómo es que está estructurado.

Algorithm 1 Algoritmo Q-Learning

```
1: Inicializar Q(s,a) arbitrariamente
2: Observar estado inicial s
3: for cada episodio do
4:     while s no es terminal do
5:         Elegir acción a desde s usando política derivada de Q (ej. ϵ-greedy)
6:         Ejecutar acción a, observar r, s'
7:         Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') - Q(s,a)]
8:         s ← s'
9:     end while
10: end for
```

El pseudocódigo, apyoado de nuevo de Watkins [16], inicia con un modelo de decisión totalmente aleatorio. El agente no conce la esencia del juego, las reglas, su entorno. Únicamente cuenta con un conjunto de acciones disponibles y su estado actual. Para su primera decisión, su método de elección es totalmente aleatorio, ya que no tiene ningún aprendizaje previo.

Luego, el agente inicia su proceso de aprendizaje durante un número limitado de episodios. Un episodio es una ejecución completa del ambiente de simulación; llegar a un estado terminal. Por ejemplo, en un juego de ajedrez, un episodio tomaría desde el estado inicial con el tablero con todas las piezas en su lugar, y acabaría con un mate o

empate. Cabe aclarar, la definición de un estado inicial y qué implica llegar a un estado terminal puede cambiar dependiendo de la aplicación del problema. Quizás, podría interesarnos estudiar un número limitado de movimientos del agente al jugar, o declarar un estado terminal como perder una primera pieza.

Habiendo iniciado el episodio, ahora el agente elige una acción a estando en s con una política derivada de Q. Luego, el agente recibe una recompensa r y se traslada a un estado subsecuente s'. Teniendo estos nuevos elementos, sucede por fin la actualización del conjunto $Q(s, a)$, en el que se da una valuación final para el futuro de realizar dicha acción, junto con las recompensas asociadas al pasado. La actualización de recompensas sigue la siguiente ecuación:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)]$$

Primeramente, se suma el valor de las recompensas actuales. Luego, se suma un monto descontado por la tasa de aprendizaje α el cual consiste en la recompensa r de la acción elegida a, más la diferencia entre la máxima recompensa posible para el siguiente estado s' eligiendo la acción que cumple dicha maximización delimitada como a', descontada por una tasa de descuento γ, menos el valor de las recompensas actuales.

En resumen, esta actualización consiste en tomar en cuenta, no solo las recompensas actualizadas, ni el valor obtenido inmediatamente, sino tomar en cuenta las recompensas que podrían conseguirse en el futuro una vez transicionando a un siguiente estado. Esto, controlado por dos factores clave: la tasa de aprendizaje α y el factor de descuento γ, encargados de controlar qué tanto valen nuestras acciones del presente, y cuánto valen las expectativas del futuro.

Este proceso de actualización se continúa hasta acabar el episodio, y

luego hasta terminar de desarrollarse todos los episodios. Así, podemos desarrollar el aprendizaje a largo plazo del agente y observar su política de acción después del entrenamiento.

1.2.3. Propiedades clave

Los algoritmos de Q-Learning tienen un par de propiedades que hacen de su entrenamiento uno muy particular. Primeramente, se entiende que Q-Learning es un algoritmo *model-free*. Se denomina como tal a cualquier modelo que no necesite conocer ni estimar el entorno o su relación con él para poder aprender. En términos más técnicos, no necesita conocer directamente una función de transición $P(s'|s,a)$ o una función de recompensa $R(s,a)$. Simplemente percibe un estado, una acción a tomar, una recompensa, y el estado al que dicha acción lo llevaría, y con base en eso aprende. Watkins y Dayans lo aclaran diciendo que este tipo de modelo puede ser usado por un agente para aprender a actuar de forma óptima sin la necesidad de crear mapas o modelos para el ambiente en el que se desarrolla [15]. Una aclaración respecto a *Q-Learning* siendo un algoritmo *model-free* es que, si bien no necesita de dichas funciones para continuar su aprendizaje, esto no significa que se desarrolle desde el principio con un conocimiento dado. Simplemente aclara que no lo necesita para iniciar y continuar su aprendizaje. Qué tanto aprenda y cuál sea su política de acción al terminar es independiente.

Por otro lado, se dice que *Q-Learning* es un modelo *off-policy*. En palabras de Szepesvári, este tipo de modelos usan dos políticas distintas: una que es evaluada y mejorada conforme al aprendizaje, y otra para generar los datos del aprendizaje [13]. Por otro lado, Sutton y Barto mencionan que la idea principal de estos modelos es que se aprende el valor óptimo de la política de forma independiente a las

acciones del agente [12]. Para entender mejor esta idea, veamos de nuevo la actualización de la función acción valor:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$$

En este caso, podemos ver cómo dicha función y por tanto la política objetivo (target policy) es actualizada con base en las acciones óptimas a tomar. Es decir, la actualización es con base en el valor esperado de tomar la acción óptima conocida. No obstante, si vemos de nuevo el algoritmo 1 donde se describen los pasos de *Q-Learning*, a la hora de elegir una acción, puede que se elija la óptima conocida al momento, o directamente, una aleatoria. A esta política se le llama política de comportamiento (behavorial policy). Esta dualidad del aprendizaje y la exploración está completamente relacionada con el dilema de exploración contra explotación; es aquí donde nace la naturaleza del aprendizaje.

A continuación, se anexa una tabla de resumen con las propiedades mencionadas anteriormente para mejor entendimiento.

Cuadro 1.2. Tipos de algoritmos en aprendizaje por refuerzo

Propiedades	**Definición**
Model-free	Aprende directamente de la experiencia sin conocer una función de política o de recompensas.
Off-policy	La política para actuar difiere de la política para aprender la acción óptima.

1.2.4. Limitaciones y desafíos de Q-Learning

Si bien, el entrenamiento de agentes con *Q-Learning* tiene muchas ventajas para el desarrollo de tareas y aprendizaje, tiene desventajas y desafíos importantes que tienen que ser considerados. Dulac-Arnold mencionaba en su artículo *Challenges of Real-World Reinforcement Learning* varios de estos puntos [4].

Uno de los desafíos que Dulac menciona es que entrenar con *sistemas reales* supone costos grandes para obtener datos de entrenamiento, resultando en muestras pequeñas de información con las que maniobrar. Por esto mismo, los sistemas tienen que tener una política de exploración mucho más conservadora, resultando en aprendizajes más limitados. Además, se menciona que trabajar con escenarios de *sistemas reales* implican espacios de acciones y estados extremadamente altos, induciendo alta dimensionalidad, lo cual afecta incluso más el punto anterior. Añadiendo a los desafíos en estos escenarios, entrenar de este modo implican realidades parcialmente observables. Es decir, hay elementos de la realidad que se intenta simular y bajo la cual entrenar que no se pueden percibir. Por ejemplo, para sistemas de recomendación no poder conocer el estado mentual de los usuarios, o en un juego de decisión no percibir la cadena de pensamientos de los demás involucrados.

Finalmente, y quizás la más relacionada con el último experimento de la tesis presentada, hay ocasiones en las que no se sabe qué es lo que realmente se quiere optimizar. Hay escenarios en los que se quiere aprender del ambiente e interactuar con él, pero sin una idea clara de qué se quiere explotar.

1.3. Función de recompensas

1.3.1. Rol en Q-Learning

Una vez explicadas las bases del aprendizaje por refuerzo y los algoritmos de *Q-Learning* podemos ahondar en el elemento central de este trabajo: la función de recompensas. Si el objetivo de los algoritmos antes mencionados es el desarrollo de un agente, la función de recompensas es el punto central que le da orientación a su aprendizaje.

El rol de la función de recompensas en *Q-Learning* es paradójico en cierto sentido. Es, por un lado, un elemento de recompensas inmediatas, pero también de alineación a largo plazo. Sutton & Barto mencionan que dicha función no nos tiene que dar conocimiento previo de qué queremos hacer. Simplemente, debemos indicarle al agente qué es bueno o malo en una recompensa inmediata. No decirle exactamente qué hacer, sino darle retroalimentación en cada decisión que tome. No obstante, aun cuando el enfoque es inmediato, y no indicativo o un set de instrucciones, son estas retroalimentaciones de momentos específicos que cambian lo que aprende a la larga nuestro agente [12].. Mencionan Sutton & Barto claramente que si uno cambia la función de recompensas, cambia por completo la tarea encomendada [12].

Es vital para llevar a cabo el aprendizaje de un agente en *Q-Learning* el tener una función de recompensas bien definida, que oriente hacia el futuro mediante la inmediatez. Creo yo, aquí nace el reto principal de desarrollar correctamente un agente.

1.3.2. Diseño de funciones de recompensas

El diseñar las funciones de recompensas, punto central de esta tesis, es un aspecto esencial para poder entrenar correctamente un agente. Ibrahim Sinan lo define como el proceso intricado y matizado que implica alinea una función de recompensas con metas esperadas y un comportamiento deseado. Además, separa el diseño de recompensas en dos conceptos que son un eje central para esta tesis[11]:

- **Ingeniería de Recompensas**: implica la creación de la función de recompensas en sí misma.
- **Modelado de recompensas**: hacer modificaciones a las recompensas de la función ya definida para mejorar el rumbo del entrenamiento.

En primer lugar, la ingeniería de recompensas conlleva no solo la programación para traducir a código las puntuaciones de distintas acciones, sino el conocimiento necesario para basar dichas recompensas en un soporte teórico del problema en cuestión. Por ejemplo, yo puedo saber cómo crear una función en Python que asigne 10 puntos a un agente de ajedrez por comer con un peón hacia afuera, pero eso no significa que dicha recompensa tenga una base teórica fuerte que pueda guiar a mi agente a aprender a jugar correctamente.

También, detalla Sinan, la ingeniería de recompensas debe conseguir un balance entre ser lo suficientemente informativo como para facilitar el aprendizaje, y no ser demasiado informativo para evitar soluciones triviales. Una vez más, el aprendizaje por refuerzo nos presenta un dilema al que dar una respuesta para entrenar agentes de forma correcta [11].

Por otro lado, el modelado de recompensas es la forma en la que guiamos en aprendizaje de nuestro agente más allá de todo lo que implican los procesos de decisión de Markov [9]. En este caso, debe haber un rastreo del desempeño del agente por cada entrenamiento para saber qué recompensas modificar: ¿hay algo que mi agente esté ignorando? ¿está atascado en una cadena de decisiones?

Nuevamente, tanto la ingeniería de recompensas como el modelado de recompensas son ejes centrales de lo que se busca conseguir con esta tesis. Es importante para futuras secciones entender cuál es el aporte de usar LLMs para diseñar las funciones. Entender correctamente si aporta directamente a la creación de funciones apoyada en conocimiento de la materia, y si mejora la facilidad de modelar recompensas interactuar mediante lenguaje natural para su cambio.

1.4. Modelos Extensos de Lenguaje (LLMs)

1.4.1. Definición

Los modelos extensos de lenguaje (o *LLMs*) son modelos de probabilidad autorregresivos basados en *transformers* que, al ser entrenados con bases enormes de texto y escalar sus parámetros, predicen el siguiente fragmento de una secuencia de texto. Como se descubrió por Brown et al., estos modelos tienen un gran potencial y habilidad de aprender contexto para resolver solicitudes sin necesidad de ajustar parámetros o aplicar un entrenamiento a un caso de trabajo específico [3].

Preguntar cuál es la utilidad de los modelos extensos de lenguaje es pertinente, pero al mismo tiempo, de una muy amplia respuesta. Regresando a Brown, estos modelos se benefician de no ser ajustados y

aprovecharse de poca información de entrada post entrenamiento para resolver diferentes tareas. Esta ventaja de los *LLMs* fue lo que llevó a Bommasani et al. a catalogarlos como modelos de fundación, ya que por la escala de su entrenamiento y su adaptabilidad lo permiten generalizarse a múltiples escenarios [2]. Es en esta misma característica donde una gran parte de esta tesis recae. Los modelos extensos de lenguaje pueden ayudarnos a resolver desde casos clásicos de procesamiento de languaje natural, hasta calificar movimientos de conecta 4. Hasta dónde llega la aplicación de estos modelos, dónde está su límite, y qué tan sobre usado es en la actualidad, son preguntas extremadamente interesantes. No obstante, estos cuestionamientos salen de los límites de lo explorado en esta tesis.

1.4.2. Origen y evolución

El origen de los modelos extensos de lenguaje podría datarse a 2013, cuando Mikolov et al. introdujeron una representación de palabras convertidas en un vector a la que se le llamó *Word2Vec*. Esto, daba pie a un avance en el área de procesamiento del lenguaje natural. Ya no solo para analizar la distribución de palabras, sino que, al convertirse a espacios numéricos, se podían hacer representaciones para inferir relaciones sintácticas y semánticas entre palabras. Entre los hallazgos principales que permitirían futuros avances, está el que las palabras que ocurren en contextos simiares tienden a tener significados similares [8].

Posteriormente, Peters et al. llevaron más allá las habilidades de *Word2Vec*. Encontraron que su mayor limitación era que este tipo de enfoques realizaban representaciones de palabras singulares. No obstante, no podían manejar que una palabra llegara a tener diferentes significados. Sí, podían hacer combinaciones semánticas de palabras

singulares, pero no dar fe al que una misma palabra pueda ser totalmente distinta dependiendo del contexto en el que se use. Así, se aprovecharon de la inovación de los LSTMs (redes neuronales recurrentes con memoria a corto y largo plazo) para conjuntarlo con modelos de lenguaje bidireccionales y obtener más información del contexto usado para una palabra [10]. A esta innovación de Peters se le llamó *ELMo*.

Luego, Vaswani et al. hicieron lo mismo que Peters en su momento con Mikolov: se apoyaron del avance en el área y trataron de mejorarlo. En este caso, notaron un gran problema en el exceso de cómputo y la necesidad de secuencialidad en las tareas de tecnologías como *ELMo*. Y, en 2017, introdujeron al mundo los *transformers*. Proponían una arquitectura de red simple basada en mecanismos de atención que se deshacen de la recurrencia y las convoluciones [14]. El modo en que funcionan es usando tanto un *encoder* como un *decoder*. Por un lado, se trabaja el texto de entrada al que se le convierten las palabras a números con *embeddings*, luego se le asignan pesos a las palabras para entender su significado y posición, y finalmente, el uso de *self attention*. Este último, una enorme innovación en el mundo del aprendizaje de máquina, permite a los *transformers* modelar dependencias sin importar dónde se encuentren en la secuencia de palabras y qué tan lejano esté. Para lograrlo, se calculan las relaciones entre cada una de las palabras contra las demás, y hasta con una misma. Y, para alzar incluso más el logro, podían realizar todas estas tareas con cómputo paralelo, al no necesitar cada palabra de ser tratada de forma secuencial para llegar al resultado final.

Es aquí, finalmente, donde en 2020 Brown y colegas de OpenAI introducen ante el mundo los experimentos realizados con GPT-3. Dichos autores se dieron cuenta por estudios pasados que, al escalar el

orden de magnitud de *transformers* con un solo entrenamiento y sin información adicional, habría posibilidad de conseguir resultados inimaginables en el campo de procesamiento de languaje natural. Así, presentaron un modelo de lenguaje autorregresivo con mas de 175 mil millones de parámetros, entrenados bajo una base de datos de texto conformada por *CommonCrawl*, páginas de *Wikipedia*, libros digitales, y múltiples corpus de texto de libre acceso. De este modo, Brown et al. descubrieron que al aumentar la escala de dichos modelos de lenguaje sin requierimientos de ejemplos contextuales (*few shot learning*) servía para resolver una gran variedad de tareas [3].

Posterior a la introducción de GPT-3, Bommasani et al. propusieron el concepto de modelos de fundamento, en donde entran los modelos extensos de lenguaje. Su característica principal: la adaptabilidad y capacidad de resolver tareas emergentes. En palabras de Bommasani, son *centrales pero incompletos*.

Desde [2] hasta la actualidad, GPT ha llegado hasta su quinta versión. Distintas alternativas a OpenAI han emergido, y el aprovechamiento de esta tecnología puede decirse ha sido usada para resolver una variedad de tareas que, probablemente, ni Bommasani ni Brown pensarían que llegara tan lejos. La llegada de estos modelos extensos de lenguaje, por más cansado que esté el argumento, se ha consolidado como uno de los avances más contundentes de la inteligencia artificial moderna.

1.5. Trabajos relacionados

1.5.1. Relación entre aprendizaje por refuerzo y LLMs

En los últimos años, han habido varias áreas de trabajo en la que los LLMs y el aprendizaje por refuerzo llegan a conectarse. Al fin y al cabo, el aprendizaje por refuerzo busca entrenar agentes mediante recompensas, y los LLMs tienen la característica de ser generalizables.

Entre las áreas de conexión de ambos tenemos Aprendizaje por Refuerzo por Retroalimentación de Experiencia (RLEF por sus siglas en inglés). Consiste en reducir recompensas manuales o explícitas a conjuntos de palabras que evalúan el rendimiento del agente [1]. En este sentido, mi tesis actual comparte el objetivo de intentar cerrar la brecha entre la experiencia y razonamiento humano y la optimización del aprendizaje de máquina. No obstante, marca la diferencia al ser las palabras mismas las que orientan a un LLM a poder calificar, y, así, orientar el entrenamiento de un agente.

Por otro lado, el entrenamiento de agentes basados en LLMs es altamente utilizado en ambientes de texto. Mencionan Radmehr et al. que los LLMs pueden servir tanto de evaluadores como de políticas mismas. Más aún, encuentran que los agentes basados en texto pueden aprender más que agentes tradicionales en contextos donde el uso de texto es predominante [**radmehr024**]. El trabajo que presento comparte la idea de la oportunidad de LLMs como evaluadores, pero se distingue al no ser agentes de lenguaje natural para ambientes de texto, sino que usar los modelos de lenguaje para evaluar ambientes de juegos generalizados.

1.5.2. LLMs para diseño de recompensas

Antes de hacer la tesis, y desde la introducción de GPT a modo de *API*, tenía la firme idea de poner a prueba la idea que aquí presento como trabajo de titulación. Inicialmente, contemplaba que mi trabajo no tenía todavía una aportación en la literatura similar. Pero, en 2023, Kwon y colegas de Cambridge postularon el trabajo de investigación *Reward Design with Language Models*. De forma definitiva, es este trabajo el más relacionado a la hipótesis que aquí presento. Aun así, guardan diferencias importantes que hacen de la tesis aquí presentada una continuación a la experimentación inicial de Kwon.

Dicho trabajo consiste en usar un LLM como una función proxy o intermediaria entre la definición de la función y la evaluación. El LLM obtiene una descripción del estado, las acciones, y las recopmensas recibidas, y aprovecha para dar una retroalimentación [6]. De este modo, la función de recompensas es modificada explícitamente para continuar un entrenamiento de aprendizaje por refuerzo tradicional. Es decir, no es el LLM el que devuelve los valores, sino el que asesora u orienta el entrenamiento para ser modificado.

Además de la diferencia funcional de la propuesta de ambos trabajos, existe un contraste en el tipo de tareas que cubren Kwon et al. En su caso, se decantan por aplicar su algoritmo a escenarios donde el texto es predominante. Específicamente, prueban su hipotesis en juegos de negociación con pocas acciones para poder cambiar con mayor facilidad la función.

En resumen, si bien el aporte de Kwon y el mío vienen de ideas sumamente similares y comparten el uso de lenguaje natural para orientar el aprendizaje, las mayores diferencias son que para Kwon, el modelo es un asistente de la función del agente, mientras que en el

trabajo aquí presentado es uno mismo quien orienta al LLM para devolver directamente los valores numéricos resultantes de evaluar episodios y acciones dada una descripción del juego. Uno funge como evaluador de una función, otro toma el rol de la función misma. Además, el tipo de experimentos difieren notablemente.

Capítulo 2

Hipótesis y Objetivos

Resumen del capítulo.

- Hipótesis central
- Objetivo general
- Objetivos específicos

2.1. Primer título del capítulo

2.1.1. Subtítulo

aaaaaaaa

Capítulo 3

Metodología

En este capítulo se explican los diferentes experimentos realizados para contrastar el uso de LLMs como funciones de recompensas contra entrenamientos estándar usando valores numéricos asignados manualmente.

3.1. Propósito

El objetivo de realizar los experimentos que se mencionarán a continuación es comprobar si un modelo extenso de lenguaje (LLM) puede tener resultados similares y un rendimiento notable al ser usado como función de recompensas para el entrenamiento de agentes en distintos escenarios. Es importante aclarar que no se busca que la propuesta tenga en general un mejor rendimiento que su versión estándar, sino que pueda competir contra esta para posicionarse como una alternativa que ofrezca un entrenamiento guiado por lenguaje natural.

Los experimentos tienen un enfoque meramente experimental y comparativo. Se busca comparar ambos algoritmos en los mismos juegos y contrastar diferentes ámbitos delos agentes: porcentaje de victorias, recompensas acumuladas, cómputo, costo, y su desempeño en pruebas post-entrenamiento.

Es importante para la fase experimental no solo el rendimiento numérico, sino la experiencia del usuario. Esto debido a que se busca el usuario pueda comunicar a la función de recompensas con lenguaje natural sus preferencias de entrenamiento para facilitar la codificación del algoritmo y el ajuste de las recompensas.

3.2. Entornos de experimentación

Para elegir los entornos donde se pondrían a prueba tanto el algoritmo propuesto como su contraparte en su versión tradicional fueron los siguientes: *Tic-Tac-Toe* y *Frozen Lake*. A continuación explicaremos cada uno a detalle junto a la justificación de su elección.

3.2.1. Tic-Tac-Toe

El primer entorno considerado para la experimentación es el juego de dos jugadores *Gato* o mejor conocido como *Tic-Tac-Toe*. Consiste en un tablero de 3x3 posiciones donde cada uno de los dos jugadores posiciona una ficha de forma alternada por turnos. El primer jugador que concatene tres fichas en una fila, columna o diagonal gana la partida. Las condiciones de terminación de una partida (o episodio) del juego es que uno de los dos jugadores gane, o que todos los espacios del tablero hayan sido utilizados sin una victoria para terminar en empate.

Tic-Tac-Toe tiene la característica de ser un juego determinista y discreto. Es decir, el espacio de acciones es discreto y no existe variabilidad estocástica al realizar una misma acción dado un estado. Siempre que los agentes elijan posicionar una ficha en una posición desocupada sucederá exactamente lo mismo. Otra particularidad del juego es que la cantidad de estados distintas en el juego es relativamente pequeño. Específicamente, existen únicamente 138 posiciones terminales que llevan a una victoria.

Una cuestión importante a tratar con juegos por turnos al entrenar un agente es: ¿cuál será su oponente a lo largo del entrenamiento? En este caso, es un algoritmo de elección aleatoria. Su funcionamiento consiste en identificar las posiciones vacías en el tablero y colocar una ficha al ser su turno de jugar. Llegaron a haber pruebas en el juego análogo de conecta cuatro de entrenar contra sí mismo, pero los resultados, mostrados en la sección posterior, fueron poco contundentes para ambos algoritmos.

La motivación de la elección de *Tic-Tac-Toe* como primer experimento tiene varias explicaciones. Primeramente, sirve como un punto de partida poco complejo al presentar relativamente pocos

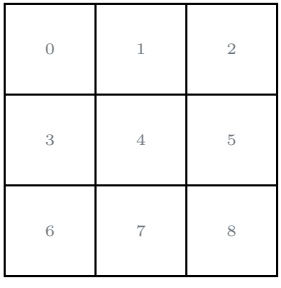
estados, y contener en sí mismo un número de acciones bastante bajo. De este modo, nos ayuda a ser un primer paso antes de pasar a un entorno más complejo. Esto facilita el tener un número limitado de llamadas a los LLMs y reducir costos. Por otro lado, presenta la variante de juegos que involucran a más de un agente, por lo cual se puede explorar cómo rinden los algoritmos de aprendizaje por refuerzo en dicho campo. También, cuenta con la gran ventaja de ser determinístico y no tener ruido estocástico, para poder analizar las comparaciones en condiciones lo más similares posibles.

Por la parte de la codificación del juego, creé una clase del entorno en un ejecutable de *Python*. Esta simplemente marca los movimientos disponibles en el tablero y si se ha llegado a una posición ganadora. Para poder definir el tablero y que el agente interactúe con él, empleé un vector de posiciones que representa los 9 espacios en el tablero.

$$s = [\,' ', 'X', 'O', ' ', ' ', 'X', ' ', 'O', ' '\,]$$

A su vez, cada índice del vector corresponde en orden a las posiciones del tablero, empezando desde arriba a la izquierda y moviéndose hacia la derecha hasta completar las tres filas, como se muestra en la siguiente figura.

Figura 3.1. Vector de posiciones de Tic-Tac-Toe



3.2.2. Frozen Lake

Nuestro segundo entorno de ejecución es un juego de un solo agente llamado *Frozen Lake*. Este, consiste en controlar a un elfo en un tablero de 4x4 celdas para llegar desde su punto de inicio hacia una meta. Lo que hace complicada esta tarea es que, a lo largo del tablero hay múltiples celdas denominadas como hoyos, donde si el agente decide posicionarse terminará el episodio en derrota.

Su clase, función de recompensas e interfaz gráfica se consiguió del paquete de *Python* conocido como *Gymnasium*, popular en el mundo del aprendizaje por refuerzo por tener múltiples juegos con funciones de recompensas ya probadas y mucho apoyo en documentación.

A continuación, mostramos una imagen de la interfaz gráfica para dar una mejor idea de qué es el juego y cómo funciona.

Figura 3.2. Tablero de *Frozen Lake* en un estado aleatorio.

La celda de la esquina superior izquierda representa el punto de partida del agente. Por otro lado, la inferior derecha donde puede

apreciarse una caja de regalo representa la meta donde debe llegar el elfo para terminar la partida en victoria. El agente, en el estado que se representa, se encuentra una celda abajo del punto de partida (representado por el elfo). Finalmente, las celdas navegables son las que no están ocupadas por hoyos, representadas por los círculos azules. El espacio de acciones del agente consiste en cuatro movimientos distintos: $\mathcal{A} = \{\texttt{LEFT, RIGHT, UP, DOWN}\}$. Partiendo desde el inicio, el agente tiene que moverse celda por celda en el tablero con estas cuatro acciones (no puede decidir no moverse).

Una característica importante de este segundo experimento es que permite cambiar la posición de los hoyos en un mapa por cada episodio. Los algoritmos de aprendizaje por refuerzo estándar y el propuesto se probaron tanto entrenando para un solo mapa, como cambiando mapas por cada episodio.

Comparando con el primer entorno de ejecución, *Frozen Lake* comparte el ser un juego determinístico en sus acciones (al menos, en la versión utilizada) y tener un espacio discreto de acciones y estados. Luego, difiere de *Tic-Tac-Toe* en solo necesitar de un jugador, y competir contra una dinámica de entorno más que contra otro agente. Esto nos ayuda a que la experiencia del contrincante no se anteponga a probar el rendimiento del agente post-entrenamiento. Además, la dinámica de cambiar tableros en el entrenamiento nos permite ver la generalización de un agente que no solo memoriza un tablero particular, sino que comprende las reglas del juego.

Así, este juego es clave para poder ver un aprendizaje generalizado y aislado del ruido de un contrincante ante el agente. El tipo de entorno permite que el aprendizaje se pueda apoyar incluso mejor en retroalimentaciones basadas en lenguaje natural para orientar qué tipo de agente queremos, núcleo de esta tesis.

3.3. Algoritmos de aprendizaje

3.3.1. Q-learning tabular

aaa

3.3.2. Deep Q-Network

aaa

Capítulo 4

Resultados

15 páginas de extensión.

- Resultados obtenidos en cada ambiente
- Comparación visual (curvas, gráficas) entre tu modelo y el estándar
- Observaciones destacadas: convergencia, sesgos, comportamiento

4.1. Primer título del capítulo

4.1.1. Subtítulo

aaaaaaaa

Capítulo 5

Discusión

15 páginas de extensión.

- ¿Qué implican tus resultados?
- ¿Cuándo conviene usar LLMs para recompensas?
- ¿Qué limitaciones encontraste? (Costo, velocidad, overfitting, etc.)
- ¿Qué cosas podrían mejorar?

5.1. Primer título del capítulo

5.1.1. Subtítulo

aaaaaaaa

Capítulo 6

Sección de apoyo para Overleaf

Este capítulo está dividido en dos secciones.

6.1. Tablas y ecuaciones

6.1.1. Hechos estilizados

$$Y = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \beta_3 X_{i3} + \beta_4 X_{i4} + \varepsilon, \tag{6.1}$$

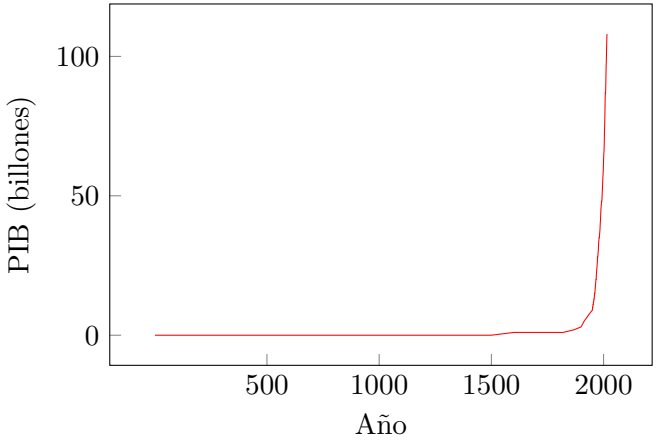
Cuadro 6.1. Resultados del modelo econométrico

	Estimador	Desv. est.	Valor t	$\Pr(>\|t\|)$
α	x	x	x	x
CI	x	x	x	x
IS	x	x	x	x
IR	x	x	x	x
PT	x	x	x	x
Error est. de res. = x con x gr. de libertad				
$R^2 =$ x y $\bar{R}^2 =$ x				
Estadístico $F =$ x, con un valor $p =$ x				

6.2. Gráficas e imágenes

6.2.1. Hechos estilizados

Figura 6.1. PIB mundial de los últimos dos milenios



Fuente: elaboración propia con datos de Maddison (2010) y el Banco Mundial.

6.3. Algoritmos

6.3.1. Hechos estilizados

6.4. Citas

6.4.1. Hechos estilizados

Texto [1].

Algorithm 2 Algoritmo Q-Learning

Inicializar Q(s,a) arbitrariamente
Observar estado inicial s
for cada episodio **do**
    while s no es terminal **do**
        Elegir acción a desde s usando política derivada de Q (ej. ϵ-greedy)
        Ejecutar acción a, observar r, s'
        Q(s,a) $\leftarrow$ Q(s,a) + α[r + γ max$_{a'}$ Q(s',a') - Q(s,a)]
        s $\leftarrow$ s'
    end while
end for

Algorithm 3 Q-Learning con LLM para Valoración de Estados

Inicializar Q(s,a) arbitrariamente
Inicializar LLM para valoración de estados
Observar estado inicial s
for cada episodio **do**
    while s no es terminal **do**
        Elegir acción a desde s usando política derivada de Q (ej. ϵ-greedy)
        Ejecutar acción a, observar s'
        Describir s' al LLM
        Obtener valoración r del LLM para s'
        Q(s,a) $\leftarrow$ Q(s,a) + α[r + γ max$_{a'}$ Q(s',a') - Q(s,a)]
        s $\leftarrow$ s'
    end while
end for

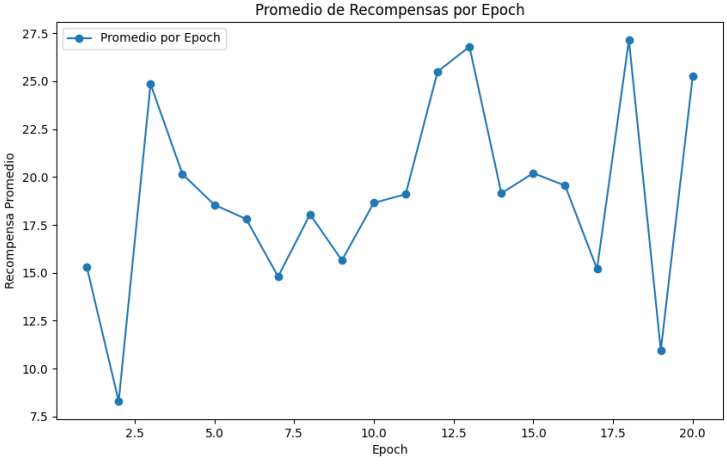


Figura 6.2. Resultados de Shopping List con algoritmo estándar.

Conclusiones

- ¿Se cumplió tu hipótesis?
- ¿Qué aprendiste del proceso?
- ¿Qué aplicaciones reales podría tener tu modelo?

Diez páginas de extensión.

Bibliografía

[1] T. Atashbar. *Reinforcement Learning from Experience Feedback: Application to Economic Policy*. Inf. téc. International Monetary Fund, 2024.

[2] Rishi Bommasani, Drew A. Hudson et al. *On the Opportunities and Risks of Foundation Models*. Inf. téc. Stanford Institute for Human-Centered AI, 2021.

[3] Tom B. Brown et al. "Language Models are Few-Shot Learners". En: *NeurIPS* 33 (2020).

[4] G. Dulac-Arnold, D. Mankowitz y T. Hester. "Challenges of real-world reinforcement learning: definitions, benchmarks and analysis". En: *Machine Learning* (2021).

[5] L. P. Kaelbling, M. L. Littman y A. W. Moore. "Reinforcement Learning: A Survey". En: *Journal of Artificial Intelligence Research* 4 (1996), págs. 237-285.

[6] M. Kwon et al. "Reward design with language models". En: *ICLR*. 2023.

[7] Tor Lattimore y Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

[8] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. En: *International Conference on Learning Representations (ICLR)*. 2013. URL: https://arxiv.org/abs/1301.3781.

[9] Andrew Y. Ng, Daishi Harada y Stuart Russell. “Policy invariance under reward transformations: Theory and application to reward shaping”. En: *Proceedings of the Sixteenth International Conference on Machine Learning (ICML)*. 1999.

[10] Matthew E. Peters et al. “Deep contextualized word representations”. En: *Proceedings of NAACL-HLT*. 2018.

[11] I. Sinan. “Comprehensive Overview of Reward Engineering and Shaping in Advancing Reinforcement Learning Applications”. En: *arXiv preprint* (2024). arXiv: 2401.01098 [cs.LG].

[12] R. S. Sutton y A. G. Barto. *Reinforcement Learning: An Introduction*. 2nd. MIT Press, 2018.

[13] Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Vol. 4. Synthesis Lectures on Artificial Intelligence and Machine Learning 1. Morgan y Claypool Publishers, 2010. DOI: 10.2200/S00268ED1V01Y201005AIM009.

[14] Ashish Vaswani et al. “Attention is All You Need”. En: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[15] Christopher J.C.H. Watkins y Peter Dayan. “Q-learning”. En: *Machine Learning* 8.3-4 (1992). DOI: 10.1007/BF00992698.

[16] Christopher John Cornish Hellaby Watkins. “Learning from Delayed Rewards”. Tesis doct. King’s College, Cambridge, 1989.

Diseño de funciones de recompensas
para Q-Learning
usando Modelos Extensos de Lenguaje
Escrito por Javier Nieto,
se terminó de imprimir en MES
en __.
--,
Ciudad de México.