PROYECTOS DEL CURSO TUTORIAL DE REINFORCEMENT LEARNING

MAURICIO VELASCO

Este documento propone cuatro proyectos de investigación en el área de reinforcement learning que los estudiantes desarrollarán durante el semestre. Cada proyecto tiene un responsable (escogidos por su experiencia previa en el área). Los proyectos tienen las siguientes reglas:

(1) Cada estudiante debe participar en exáctamente dos proyectos (deben informarme en cuales van a participar el primer día de clases).
(2) Los proyectos tendrán tres entregas:
   (a) Entrega 1: Semana 4
   (b) Entrega 2: Semana 8
   (c) Entrega 3: Semana 16

   En cada una de ellas el responsable del grupo deberá entregar código y preparar un demo de 20 minutos compartiendo con todos el progreso del código.

La última entrega requiere:

(1) El código del programa terminado.
(2) Un full-text paper que cumpla las reglas de submission de la conferencia ICLR.

Uno de los objetivos del semestre es que cada grupo tenga resultados como para ser enviados a la conferencia

https://waset.org/learning-representations-conference-in-august-2023-in-sydney

como abstracts o full-text papers (ver la lista de Selected papers para ver el formato de los mismos (notar que son cortos y citan MUUCHOS artículos)). El deadline es Julio 1,2022.

1. Graph coloring (Responsable: Nicolás Castro na.castro976@uniandes.edu.co)

Hay una tradición extendida de usar reinforcement learning para construir heurísticas buenas para problemas difíciles de optimización combinatoria (ver [?B, Sección 1.3.2]). La idea general es que un problema de esta naturaleza se puede pensar como una serie de decisiones sucesivas y por ello puede resolverse como una instancia de programación dinámica usual. El problema es que para cualquier caso de interés el número de estados posibles crece exponencialmente asi que la solución exacta mediante PD no es viable.

Un acercamiento posible es entonces usar la técnica de *Rollout* [?B, Sección 2.4]. En ella se empieza con una política subóptima de base y se mejora esta política mediante look-ahead de pocos pasos y utilizar la política base al final [?B, Example 2.4.1].Tambien hay que leer el apéndice http://www.mit.edu/~dimitrib/

LessonsfromAlphazero.pdf donde se discuten variaciones de roll-out adaptativo,por ejemplo con muchas heurísticas distintas de base.

En este proyecto queremos aplicar esta idea para construir un programa que sea muy bueno produciendo coloraciones explícitas del número cromático de un grafo G con bastantes vértices (digamos varios miles). Es decir, dado un grafo G y el algoritmo debe intentar consturir una coloración de G usando un conjunto de pocos colores $[c]$ (esto es una asignación de elementos de $[c]$ a los vértices de tal maner que dos vértices adyacentes no tengan el mismo color). Más concretamente los objetivos del proyecto son:

(1) Escribir el problema de coloración como un problema de programación dinámica.
(2) Como base policy se puede usar greedy coloring https://en.wikipedia.org/wiki/Greedy_coloring pero hay muchas heurísticas conocidas para colorear un grafo. (ver "Using multiple base heuristics" en el apéndice del texto de Bertsekas de Lessons from alpha-go).
(3) El improvement se puede hacer mediante rollout pero podría haber tambien muchas otras opciones.
(4) Hay que probar la implementación con grafos pequeños y visualizar lo que se obtiene (en particular para buscar errores). Usar la librería https://networkx.org/. En particular esta tiene muchos graph generators, ideales para probar si lo que estamos haciendo funciona https://networkx.org/documentation/stable/reference/generators.html. Los ejemplos son clave!
(5) Para poder decir que el algoritmo que encontramos es bueno, es necesario compararlo con otras opciones. Un método alternativo que a veces funciona muy bien es *simulated annealing*. Buscar referencias sobre esto e implementarlo para el problema de coloración de grafos. Comparar este procedimiento con el que sale de Rollout.
(6) Aplicación: El problema de asignación de horarios a examenes finales de la Universidad puede considerarse como una instancia de este tipo. Quizás mediante reinforcement learning seamos capaces de resolverlo (probablemente podamos pedir datos a registro si quisiéramos probar en la práctica).
(7) Qué otras aplicaciones se les ocurren de poder colorear grafos grandes con pocos colores?

2. Graph Neural Networks (GNNs) y reinforcement learning en juegos (Responsable: Federico Galvez f.galvez@uniandes.edu.co)

El propósito del proyecto es crear una inteligencia artificial que aprenda a jugar a las damas. Este es un juego muy conocido, con reglas muy simples que lleva a situaciones estratégicas complejas (usaremos las reglas de https://en.wikipedia.org/wiki/Draughts, pero ver variantes https://es.wikipedia.org/wiki/Damas#Damas_peruanas). Nos servirá para aprender acerca de RL en juegos y para explorar algunas ideas de investigación sobre el rol de GNNs en reinforcement learning.

Quiero empezar por describir algunas ideas claves para el proyecto:

(1) La excelente charla https://www.youtube.com/watch?v=WQS7933ub9s (sobre su libro https://web.mit.edu/dimitrib/www/LessonsfromAlphazero.pdf) explica que hay dos algoritmos esenciales cuya combinación determina

el éxito de alpha-zero: Offline learning y Online play. Nuestro objetivo es desarrollar ambos para eljuego de Damas. **Convenciones:** Los dos jugadores se llamarán AZUL y ROJO y nosotros entrenaremos al jugador AZUL, mientras que el oponente será ROJO.

(2) El **Offline-learning** tiene por objetivo estimar una función de valor para nosotros (AZUL). Más concretamente esta función de valor debe asignar a cada estado s posible (s consiste de un vector con tantas componentes como casillas válidas en el tablero y valores en $\{-1, -0.5, 0, 0.5, 1\}$ que describen como esta ocupada cada casilla y un bit adicional que dice de quién es el turno) una valoración de qué tan probable es que azúl gane a partir de s. Queremos implementar tal aproximación mediante un Graph Neural Network (ver la excelente charla de Ribeiro de acá https://youtu.be/lddx3mvLSJw?t=3100 para entender qué son y https://tkipf.github.io/graph-convolutional-networks/ para implementación). En nuestro caso los vértices del grafo serán las casillas válidas del tablero en damas y dos casillas serán consideradas adyacentes si se puede llegar de una a la otra mediante un paso diagonal. El estado del tablero puede pensarse como una función en los vértices del grafo (qué asigna a cada vértice un valor que determina cómo esta ocupado ese vértice). El valor del tablero será el output de un GNN en G con input la configuración (preguntarme detalles después de very entender la charla de arriba). El entrenamiento de esta red es un proceso iterativo (ver el excelente http://www.gm.fh-koeln.de/ciopwebpub/Kone15c.d/TR-TDgame_EN.pdf y el regular https://www.researchgate.net/publication/242405861_Reinforcement_learning_project_AI_Checkers_Player para el caso específico de las damas),en resumen se hace así:
   (a) Como estrategia inicial de AZUL se usará:
      - Una jugada válida, escogida aleatoriamente entre las opciones ($Unif - Random$), si V aún no esta definida.
      - one-step lookahead con la función V dada, una vez tangamos definida a V
   (b) Usando la estrategia del item anterior para AZUL y $Unif - Random$ como estrategia de ROJO simulamos muchos partidos (algunos miles), que llamaremos historias de juego. Cada turno de azul en una de esas historias es un Data point.
   (c) Usaremos el algoritmo Temporal Differences $TD(\lambda)$ para estimar lo que debería valer $V(s_t)$, llamemos a ese estimado $\overline{V(s_t)}$ para cada uno de esos data points usando los daos que tenemos.
   (d) Entrenamos el GNN para que minimice el error $\sum_t \|F(s_t) - \overline{V(s_t)}\|^2$ donde F es la función determinada por nuestro GNN.
   (e) Nuestra nueva V es la F que resulte del entrenamiento de la GNN. Este proceso se puede repetir algunas veces. Notar que es MUY time-consuming asi que sería muy interesante paralelizarlo (hablar con Mauricio Morales en facultad de ciencias cuando ya tengamos un plan, https://www.machinelearningplus.com/python/parallel-processing-python/, pensar en esto...).

(1) El **Online play** se calcula mediante multi-step lookahead usando la función V que estimamos en el punto anterior. Esperamos que este jugador sea MUCHO mejor que el que se obtiene sólamente mediante offline learning.

El output principal del proyecto deben ser gráficas de aprendizaje, en las que podamos ver cómo cambia la probabilidad de ganarle a ROJO cuando este utiliza la estrategia de escoger una jugada aleatoriamente de manera uniforme. Lo interesante es ver cómo varia esta curva mientras variamos los muchos parámetros del algoritmo: -el número de iteraciones del offline learning, el número de pasos de look-ahead del online learning, la arquitectura del NN, el número de episodios de entrenamiento.

2.1. **Entregas.**

2.1.1. *Entrega 1: clases generales y jugadores uniformes.*

- class VectorEstado: Un VectorEstado recuerda la configuración de piezas en el tablero (un vector de tamaño $n^2/2$ con entradas $\{-1, -0.5, 0, 0.5, 1\}$ y una componente adicional que recuerda de quién es el turno (aclarar si se quiere representación pre o post jugada, detalles en http://www.gm.fh-koeln.de/ciopwebpub/Kone15c.d/TR-TDgame_EN.pdf)
  - VectorEstado.visualizar(): es importante poder ver la configuración del tablero en cada instante (una solución fácil sería heatmaps con matrices como en).
- class TransicionesEstado
  - TransicionesEstado.esvalido(s,s'): dados dos VectorEstado s y s' decide si pasar de uno al otro en un turno es algo posible según las reglas del juego.
  - TransicionesEstado.movimientosPosiblesPieza(s,p): dado un VectorEstado s y una posición p (un entero) ocupada por el jugador en turno lista todas las posiciones posibles para esa pieza en un turno según las reglas del juego. Retorna una lista de VectorEstados.
  - TransicionesEstado.movimientosPosibles(s): Dado un VectorEstado s retorna la lista de VectorEstados de todas las jugadas admisibles en un turno iniciando en s.
- class JugadorDAMAS: Esta es una clase abstracta que representa un jugador cualquiera. Todas las estrategias de juego se implementaran como subclases del JugadorDAMAS.
  - JugadorDAMAS.nextMove(s): dado un VectorEstado s que sea turno del jugador retorna el nuevo estado s' en el que el juego quedaría despues del turno del jugador (la lógica de como se elige el nextMove depende del jugador concreto que se instancie.
- class JugadorUnforme::JugadorDAMAS: Esta subclase corresponde a un jugador que elige su jugada de manera aleatoria uniforme entre las jugadas válidas disponibles (que se calcuan con la clase TransicionesEstado)
- class JuegoDAMAS: Esta clase recuerda qué JugadorDAMAS es AZUL y que JugadorDAMAS es ROJO y una vez implementado esto puede
  - JuegoDAMAS.simula(NumJugadas, VectorEstadoInicial): Permitimos un número límite de jugadas y damos una tablero inicial válido y el algoritmo corre las estrategias de los jugadores que fijamos.

  - JuegoDAMAS.partidas(Número): Retorna las historias de juego de un cierto número de partidas (es decir la sucesión de estados que costituye la partida) y los datos de cuantas partidas ganó cada jugador. Típicamente uno o ambos jugadores usarán estrategias aleatorias así que las historias de las partidas típicamente seran distintas.

El objetivo principal de la primera entrega debe ser poder correr una simulación en JuegoDAMAS en el que AMBOS jugadores escojan sus jugadas de manera uniforme entre las posiciones válidas implementando las clases descritas antes.

2.1.2. *Entrega 2: Online play.*

- class JugadorValor::JugadorDAMAS: Un jugador de Valor tiene una función de valor V que puede evaluar en todo estado s. Usando esta evaluación realiza multistep look-ahead de k pasos donde k es un entero dado.
  - JugadorValor.ValueFunction(s): evalúa la función de valor y retorna un número real.
  - JugadorValor.next(s): Primero calcula todos los árboles de posibilidades para k jugadas en el futuro. Para cada una de las hojas evalua el valor y elige la rama inicial que lo lleve a maximizar ese valor.
- ***Inicializar el estimador de la función de valor: Hay que crear el graphNN y entrenarlo con los datos que salen de partidos (de la entrega 1) para inicializar la función de valor.

  El segundo item (con ***) es el objetivo principal de la segunda entrega.

2.1.3. *Entrega 3: Offline learning y la reunión de todas las partes.* El objetivo de la entrega 3 es mejorar la estimación de la función de valor de manera iterativa usando temporal differences y graph neural networks. La idea es ver cómo mejora la capacidad de juego de nuestro agente contra el oponente uniforme en la medida en que esta estimación va mejorando. El resultado central de esta entrega deben ser gráficas que miden esto análogas a las Figuras 2,3,4 de https://www.researchgate.net/publication/242405861_Reinforcement_learning_project_AI_Checkers_Player

3. Dynamic programming and sums-of-squares for real-time motion planning (Responsable: Sergio Cristancho se.cristancho@uniandes.edu.co)

La programación dinámica permite encontrar el óptimo global de un problema (invirtiendo una cantidad considerable de cómputo) mediante la construcción de una función de valor ("cost-to-go" function). Si las condiciones del ambiente cambian esta función ya no describiría un óptimo global pero puede seguir siendo una guía muy útil que podemos "combinar" con algoritmos locales para lograr buenos resultados en ambientes nuevos. Mientras que el cálculo inicial de la función de valor requiere una gran cantidad de computo su uso para calcular trayectorias óptimas es muy eficiente si el número de controles entre los cuáles debemos elegir a cada paso no es muy grande. En este proyecto exploraremos el uso de programación dinámica combinada con ideas recientes de sumas de cuadrados en sistemas dinámicos para planear la trayectoria de un vehículo en tiempo real. El vehículo estará en un laberinto (conocido de antemano) y tendrá que llegar lo más rápidamente posible a un punto dado del mismo mientras esquiva una serie de obstáculos (que aparecerán

en tiempo real) y bajo la acción de una corriente aleatoria. Quiero empezar por describir algunas ideas claves para el proyecto:

- Programación dinámica. Dado un laberinto en el que esta marcado un punto de salida queremos calcular la función de valor ("cost-to-gofunction") del problema de salir lo más rápidamente posible del laberinto mediante el algoritmo de Dijkstra (que es programación dinámica). El objetivo sería reproducir los resultados de https://towardsdatascience.com/solving-mazes-with-python-f7a412f2493f
- Modelo de vehículo (Dubin's car model) El estado de un vehículo $2D$ está determinado por tres números (x, y, ψ) donde x, y son las coordenadas del centro de masa y ψ es el ángulo que indica la dirección "hacia adelante" del vehículo. Usaremos la dinámica dada por el modelo de Dubin con una corriente no determinada w como esta descrito en la Sección 5 del excelente paper http://www.princeton.edu/~aaa/Public/Publications/or_isos.pdf (cuyo autor viene a Medellín para MAPI2). Acá nuestro único control es la aceleración angular y el vehículo va siempre hacia adelante con velocidad v fija (más una perturbación desconocida horizontal w que asumimos en un cierto rango). Como comentario al margen las curvas trazadas por vehículos de Dubin tienen propiedades matemáticas muy interesantes (http://planning.cs.uiuc.edu/node821.html)
- Sumas de cuadrado en control: En la Sección 5 de http://www.princeton.edu/~aaa/Public/Publications/or_isos.pdf se usan algoritmos de sumas de cuadrado para escoger entre 5 controles posibles a cada instante. Se usan polinomios para construir una demostración de que el vehículo no va a chocar con niguno de los obstáculos cercanos y se escoge el primer control (entre 5 opciones) para el que tal demostración exista (acá pueden ver como funciona en tiempo real https://www.youtube.com/watch?v=J3a6v0tlsD4)
- Combinación de todas las anteriores: Ahora tenemos nuestro vehículo (Dubin's car) dentro del laberinto. El vehículo quiere llegar lo más pronto posible al punto marcado y asegurarse de no chocar con los obstáculos (notar que los obstáculos no se conocen al calcular la función de valor sino aparecen durante la parte online como en el numeral anterior). Para ello debemos escoger pocos controles (los mismos 5 que en el paper anterior) y para cada uno de ellos debemos:
  (1) Intentar, mediante DSOS/SDSOS demostrar que no chocamos contra ningun obstaculo (incluyendo los dos discos más cercanos y las paredes del laberinto). Aquellos son los controles válidos.
  (2) Entre los válidos escoger aquel que minimice la función de valor (notar que, como la perturbación w es desconocida esto debe hacerse mediante simulación).

  Para que el proyecto sea exitoso las decisiones anteriores deben poder hacerse en tiempo real asi que es fundamental el uso de DSOS-SDSOS (leer consideraciones a este respecto de la Seccion 5 y en particular usar GUROBI y no MOSEK).

Más generalmente, lo que se quiere explorar en este proyecto es la idea de separar offline learning (en este caso cálculo exacto de la función de valor, pero tambien podría ser una estimación de la función de valor con cualquier método de

reinforcement learning) de online theorem proving (que en este caso se hace con sumas de cuadrados). Esto es muy interesante porque, si nuestro modelo local es acertado, da garantías matemáticas sobre la seguridad de operación de un objeto, independientemente de cómo haya sido su entrenamiento y esto tiene una infinidad de aplicaciones (ver: http://underactuated.mit.edu/acrobot.html ó https://epubs.siam.org/doi/10.1137/16M1062569?mobileUi=0 Secciones 6,7 para ejemplos interesantes).

4. Combinatorial Multi-Armed Bandits en Green Security Games (Responsable: Nicolás Betancourt n.betancourt10@uniandes.edu.co).

La protección de áreas de conservación ante diferentes amenazas es un problema de mucho interés. Típicamente tenemos un conjunto de guardabosques que quieren patrullar un territorio protegido que esta asolado por cazadores (que instalan trampas o simplemente cazan fauna protegida en estos territorios). Trabajo reciente de [N. Betancourt et al] sugiere adaptar este contexto general a los parques naturales latinoamericanos donde se camina exclusivamente a lo largo de senderos (por la espesura y dificultad del terreno). Esto lleva a una nueva teoría de security games en grafos.

Este proyecto quiere proponer maneras de desarrollar rutinas de patrullado que sirvan para simultáneamente proteger parques y al mismo tiempo, aprender las estrategias de visita usadas por los cazadores. Un acercamiento posible involucra las ideas de reinforcement learning y más concretamente las diferentes variantes de *multi-armed bandits*. Este tipo de online learning es fundamental pues en problemas como estos típicamente no hay datos históricos disponibles. Más concretamente, este proyecto plantea mezclar las siguientes ideas:

(1) *Multi-armed bandits* El problema clásico del multi-armed bandit es el siguiente: Tenemos un presupuesto fijo M y queremos usarlo en unconjunto de máquinas tragamonedas $x_1, \ldots, x_N$ (one-armed bandits). Estas máquinas se pueden modelar como variables aleatorias con distribuciones distintas que desconocemos. La pregunta es: Cuál debería ser la estrategia que usamos para maximizar nuestro retorno total? Lo importante de un problema MAB es que hay dos fuerzas en tensión. Por un lado se quiere *explorar* (probando máquinas nuevas para estimar sus retornos) y por otro se quiere *explotar* (jugando en la máquina que, con los datos de ese momento, consideramos la mejor). Sorprendentemente es posible balancear estas dos cosas, llevando al algoritmo MAB. Para una exposición como se debe ver la clase de T. Roughgarden de acá https://www.youtube.com/watch?v=IWzErYm8Lyk sobre el multiplicative weights update, donde se explica bien qué tipo de garantías pueden obtenerse (es, en serio, un teorema difícil de creer y fácil de probar).

(2) *Combinatorial multi-armed bandits* En el caso de green security games en grafos podemos pensar que cada ciclo posible de patrullaje es una máquina tragamonedas y nuestro objetivo es maximizar el retorno (medido como haber visto a los cazadores o a sus trampas a lo largo del ciclo (si queremos retornos en $\{0, 1\}$) ó como una cantidad continua en $[0, 1]$ que mida el "valor" de las especies protegidas por patrullar el ciclo). En nuestro caso, diferentes ciclos tienen aristas en común y al seleccionar un ciclo obtenemos información tambien sobre otros ciclos, por ello sería mucho mejor

modelar el problema como un Combinatorial Multi-Armed bandit problem http://proceedings.mlr.press/v28/chen13a.pdf en el que hay arms y super-arms (creo que para nosotros arms serían las aristas y super-arms los ciclos). Cómo hacemos esto concretamente? Qué mejoras ofrece este algoritmo sobre un MAB que ignore esta estructura? Qué tipo de garantías obtenemos?

(3) *Dual-mandate patrols.* En el artículo https://arxiv.org/pdf/2009.06560.pdf se lleva a cabo la idea de usar multi-armed bandits para mejorar la protección de parques. Este artículo tiene varias ideas interesantes y sería deseable entender cómo se extienden a nuestro problema. Concretamente:
   (a) En el artículo las cosas se hacen en dos etapas. En la primera se aprenden (o estiman) los hábitos del cazador (en nuestro caso sería estimar los "valores" de las diferentes aristas del grafo) y con ello se selecciona un super-arm (mediante resover Knapsack, Seccion 4.2).Algo semejante podría hacerse acá, donde primero estimamos el valor relativo de las aristas y luego escogemos un ciclo de máximo valor.
   (b) Parten las regiones posibles de acuerdo a features semejantes. Cómo podemos hacer eso de manera no supervisada en nuestro contexto de grafos? https://arxiv.org/pdf/2006.16904.pdf.Acá el problema es más difícil pues queremos agrupar **ciclos** con features semejantes (aunque quizás basta obervar que un ciclo es un conjunto de aristas y uno debería usar métrica de wasserstein entre distribuciones de aristas, asumiendo que las aristas han sido embebidas de alguna manera (mediante Graph Embeddings, por ejemplo).
   (c) En el artículo señalan que los algoritmos MAB pueden tener un inicio muy malo, esto es letal cuando se trata de proteger un recurso como la biodiversidad, que no es fácilmente recuperable. Entender cómo hacen esto...
   (d) La sección Experiments es excelente. comparan su algoritmo con varios que ya existen y utilizan tanto datos reales como sintéticos. Hay que aprender a hacer esto, si queremos hacer contribuciones útiles...

Direcciones futuras:

- En este proyecto, como en https://arxiv.org/pdf/2009.06560.pdf, NO SE ASUME que el cazador es sofisticado y observa al guardabosques reaccionando a lo que ve, es decir, no es un Stackelberg game en el sentido preciso del término, mezclar stackelberg games con reinforcement learning me parecería interesantísimo pero no se como hacerlo... propuestas?

4.1. **Entregas.**

4.1.1. *Entrega 1: Implementación de Multi-armed bandits sencillo y listas de ciclos del grafo de la reserva.*

- Implementación del 10-armed testbed de Barto y Sutton (Sección 2.3 del libro)
- Implementación del algoritmo que lista los ciclos de la reserva. Cuántos son?
- Descripción del problema de la reserva como un combinatorial multi-armed bandits problem así http://proceedings.mlr.press/v28/chen13a.pdf.

4.1.2. *Entrega 2: Combinatorial multi armed bandits.*

- Implementación del combinatorial multi-armed bandits problema para la reserva.
- Búsqueda de bibliografía adicional reciente.

4.1.3. *Entrega 3:*

- Comparación entre la implementación nueva de la entrega anterior y otros algoritmos ya establecidos (Replicar la sección de Experiments de https://arxiv.org/pdf/2009.06560.pdf)