Deep Q-Learning multiagente aplicado a la creación de un equipo de fútbol de agentes simulados 2D

José Enrique Carrillo Pino

Orientador: Prof Dr. Dennis Barrios Aranibar

Tesis profesional presentada al Programa Profesional de Ingeniería Informática como parte de los requisitos para obtener el Título Profesional de Ingeniero Informático.

UCSP - Universidad Católica San Pablo
Noviembre de 2016

Dedico este trabajo a mis padres, por todo el esfuerzo que han hecho en sacarme adelante y a mis profesores por todas sus enseñanzas.

Abreviaturas

RL Reinforcement Learning

MDP Markov Decision Process

GPI Generalized Policy Iteration

DPG Deterministic Policy Gradient

DQN Deep Q-Network

DBN Deep Belief Network

RBM Restricted Boltzman Machine

MLP Multilayer Perceptron

CNN Convolusional Neural Network

Agradecimientos

En primer lugar deseo agradecer a mis padres por haberme brindado todo el apoyo para forjarme como un profesional.

Agradezco a la Universidad Católica San Pablo, mi *alma matter*, por haberme cobijado y brindado la formación que ahora me permitirá ayudar a construir una mejor sociedad.

Agradezco de forma muy especial a mi orientador Prof. Dr. Dennis Barrios por haberme guiado en esta tesis.

Deseo agradecer de forma especial a mis docentes a lo largo de toda mi carrera universitaria, porque fueron ejemplos que deseo seguir en mi vida profesional.

Deseo agradecer al personal administrativo de la universidad. Muchas gracias por la atención brindada y porque siempre estuvieron dispuestas a ayudarnos.

Resumen

El aprendizaje por refuerzo ha recibido un empujón gracias a la introducción de la Deep Q Network. Gracias a esto se están desarrollando varias líneas de investigación al respecto. Por otro lado, la RoboCup tiene una categoría de fútbol simulado 2D que motiva y permite probar ideas nuevas en el campo de la inteligencia artificial. En este trabajo se propone la creación de un equipo de fútbol simulado 2d utilizando el algoritmo de Deep Q Network, es desafiante e interesante porque hay que combinar este algoritmo con las técnicas de aprendizaje cooperativo de los sistemas multiagente. De esta manera se espera progresar un poco más hacia la consecución de la meta de la RoboCup que es crear un equipo robótico que juegue contra el campeón mundial de fútbol en el año 2050. Se espera que el equipo juegue al menos mejor que un equipo estandar de comparación agent2d.

Abstract

Reinforcement Learning now has more attention due to the introduction of Deep Q Network. As a result there are many investigation lines currently. In the other hand, RoboCup has a soccer simulation 2D category that stimulates and let test new ideas in the field of artificial intelligence. In this work we propose the creation of a simulated soccer team using Deep Q Network algorithm. This is interesting and challenging because we have to use cooperative learning techniques from multiagent systems as well. This way, we hope to get a little bit closer to reach the goal of RoboCup: to play with the world champion soccer team with a robotic team in 2050 and win. In this work, we hope that our team plays at least better than a standard team called agent2d.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación y Contexto

La robocup tiene como objectivo construir un equipo robótico capaz de jugar fútbol con el campeón mundial y ganar en el año 2050. Con este fin se tienen varias categorias de competición que retan a los investigadores de todo el mundo para ir mejorando poco a poco los algoritmos y técnicas utilizadas en la construcción de robots.

Una de esas categorías es la simulación de fútbol 2D, que como su nombre lo indica es una simulación. La ventaja de una simulación es que abstrae todos los detalles de construcción del hardware de los robots y permite a los investigadores centrarse en los algoritmos, en la estrategia. Gracias a esto, esta categoría sirve como una cama de pruebas muy interesante para probar algoritmos de inteligencia artificial nuevos.

Recientemente se anunció la noticia de que finalmente se había creado una inteligencia artificial capaz de ganarle al campeón mundial de Go en una ronda de 5 partidas, donde quedaron 4 contra 1. El algoritmo que logró esta hazaña combina 2 grandes ramas de la inteligencia artificial: el aprendizaje por refuerzo y el aprendizaje profundo. Con este mismo algoritmo también se creó una inteligencia artificial capaz de aprender a jugar 49 juegos de Atari teniendo como entrada solamente los píxeles de la pantalla y la puntuación.

En este trabajo se pretende dar un paso inicial hacia la creación de un equipo robotico simulado, utilizando un algoritmo del estado del arte actual, como es el aprendizaje por refuerzo profundo.

1.2. Planteamiento del Problema

El fútbol de robots simulado es un problema muy difícil que se puede modelar con un Markov Decision Process (MDP) con número de estados virtualmente infinito. El

problema que se quiere abordar en este proyecto es el aprendizaje de jugadas individuales de un agente en la modalidad de aprendizaje por refuerzo.

1.3. Objetivos

Que un agente aprenda a esquivar rivales y anotar goles usando un algoritmo de aprendizaje por refuerzo profundo.

1.3.1. Objetivos Específicos

- Modelar el problema como un MDP con sus respectivos estados, acciones y función de recompensa
- Implementar el algoritmo Deep Q-Network (DQN) en un agente y entrenarlo
- Analizar el desempeño del agente en la tarea dada

1.4. Organización de la tesis

Al final...

Capítulo 2

Marco teórico

2.1. Reinforcement Learning

El aprendizaje por refuerzo consiste en mapear situaciones a acciones, de forma que se maximice una recompensa numérica. Al agente no se le dice que acciones tomar como en muchas formas de aprendizaje de máquina, sino que debe descubrir que acciones brindan la mayor recompensa intentándolas. Es los casos más interesantes y desafiantes, las acciones pueden no sólo afectar la recompensa inmediata, sino la siguiente situación y en consecuencia, todas las recompensas siguientes. Estas dos características: búsqueda por prueba y error y recompensa retardada, son las dos características más importantes del aprendizaje por refuerzo.

Más allá del agente y el ambiente, se pueden identificar cuatro sub-elementos de un sistema de aprendizaje por refuerzo: la política, la función de recompensa, la función de valor y opcionalmente, un modelo del ambiente.

La *política* define la forma de comportarse del agente en un tiempo dado. En otras palabras, la política es el mapeamiento de un estado percibido a una acción que debe ser tomada en ese estado. Corresponde a lo que en psicología sería llamado un conjunto de reglas de estímulo-respuesta. En algunos casos la política puede ser una función simple o una tabla, mientras que en otros puede invocar computación extensiva como un proceso de búsqueda. La política es el corazón del agente de aprendizaje por refuerzo en el sentido de que por si mismo puede determinar el comportamiento. En general, las políticas suelen ser estocásticas.

Una *función de recompensa* define la meta en un problema de aprendizaje por refuerzo. Dicho de otro modo, mapea cada estado percibido (o par de estado-acción) a un número singular, la *recompensa*, indicando el valor intrínseco de dicho estado. El único propósito de un agente de aprendizaje por refuerzo es maximizar la recompensa total que recibe en el largo plazo. En un sistema biológico, no sería inapropiado identificar las recompensas con placer y dolor. De esa forma, la función de recompensa debe ser necesariamente inalterable por el agente. Sin embargo, debe servir como una base para alterar

la política. En general, las funciones de recompensa también son estocásticas.

Mientras que la función de recompensa indica que es bueno en sentido inmediato, una *función de valor* especifica que es bueno en el largo plazo. En otras palabras, el valor de un estado es el monto total de recompensa que un agente puede esperar acumular en el futuro, empezando en ese estado. Para hacer una analogía más humana, las recompensas son como el placer o el dolor mientras que los valores corresponden a un juicio más definido y con visión del futuro.

El último elemento de algunos sistemas de aprendizaje por refuerzo es un *modelo* del ambiente. Esto es algo que imita el comportamiento del ambiente. Por ejemplo, dado un estado y una acción, el modelo puede predecir el estado resultante y la siguiente recompensa. Los modelos son usados para planear, y por eso queremos decir cualquier forma de decidir un curso de acción considerando posibles situaciones futuras sin haberlas experimentado realmente.

2.1.1. El problema del aprendizaje por refuerzo

En esta sección hablamos del problema que trata de solucionar el aprendizaje por refuerzo. El problema del aprendizaje por refuerzo esta pensado como un marco de referencia sencillo para el problema de aprender a través de la interacción para lograr una meta.

El agente y el ambiente interactúan en una secuencia de pasos discretos, $t = 0, 1, 2, 3, ...$ En cada paso t, el agente recibe una representación del estado del ambiente $S_t \in S$ donde S es el conjunto de posibles estados, y en esa base selecciona una acción $A_t \in A(S_t)$, donde $A(S_t)$ es el conjunto de acciones disponibles en el estado S_t. Un paso de tiempo después, en parte como consecuencia de su acción, el agente recibe una recompensa numérica $R_{t+1} \in R$ y se encuentra a sí mismo en un nuevo estado S_{t+1}. La Figura muestra la interacción del agente con el ambiente.

En cada paso de tiempo, el agente implementa un mapeo de estados a probabilidades de seleccionar cada posible acción. Este mapeamiento es llamado la política y se denota como π_t, donde $\pi_t(a|s)$ es la probabilidad de que $A_t = a$ si $S_t = s$. Los métodos de aprendizaje por refuerzo especifican como el agente cambia su política como resultado de la experiencia. La meta del agente es maximizar el monto total de recompensa que recibe en el largo plazo.

La meta y las recompensas están íntimamente ligadas. El propósito o meta del agente esta formalizado en términos de una señal de recompensa que el ambiente le pasa al agente. En cada paso de tiempo, la recompensa es un un simple número, $R_t \in R$. Informalmente, la meta del agente es maximizar el total de recompensa que recibe. Esto no significa maximizar la recompensa inmediata, sino la recompensa acumulada en el largo plazo. Pero ¿cómo se define esto formalmente? Si la secuencia de recompensas recibidas después del paso t se denotan como $R_{t+1}, R_{t+2}, R_{t+3}, ...$, entonces ¿qué aspecto de esta secuencia queremos maximizar? En general, buscamos maximizar la recompensa acumulada, donde

el retorno G_t, esta definido como una función de la secuencia de la recompensa. En el caso más simple, el retorno está definido como la suma de las recompensas:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{2.1}$$

Este enfoque tiene sentido en aplicaciones donde hay un paso final, y las interacciones del agente se dividen en subsecuencias que llamaremos *episodios*. Sin embargo, existen casos donde las interacciones del agente continúan sin un límite, a estas tareas las llamaremos *tareas continuas*. La fórmula del retorno (2.1) es problemática para tareas continuas, ya que el retorno podría ser infinito. Para solucionar esto necesitamos un concepto adicional, el *descuento*. De acuerdo a este enfoque, el agente trata de seleccionar acciones de modo que la suma descontada de las recompensas que recibe en el futuro sea maximizada. La fórmula del *retorno descontado* es:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{2.2}$$

Donde γ es un parámetro, $0 \leq \gamma \leq 1$, llamado el *factor de descuento*.

Para unificar la visión de las tareas episódicas y continuas, podemos considerar que el final de un episodio sea la entrada a un *estado de absorción* que siempre dirige al agente al mismo estado y retorna sólo recompensas de cero. Como en el diagrama

Las señales que provienen del ambiente deben cumplir una propiedad importante: La propiedad de Markov. Esta propiedad indica que la señal de estado debe incluir las sensaciones pasadas de forma compacta, de forma que toda la información relevante es retenida. Una tarea de aprendizaje por refuerzo que satisface la propiedad de Markov se llama *proceso de decisión de Markov*, o *MDP*.

Casi todos los algoritmos de aprendizaje por refuerzo involucran estimar una *función de valor*, que en palabras sencillas significa estimar que tan buena es el estado donde se encuentra el agente, en base a la recompensas futuras que pueden ser esperadas. Pero por supuesto que las recompensas que el agente puede esperar recibir dependen de que acciones va a tomar. De este modo y es la recompensa esperada empezando en el estado s y siguiendo la política π. De forma similar, denotamos el valor de tomar la acción a en el estado s bajo una política π como $q_\pi(s, a)$.

Una propiedad fundamental de las funciones de valor es la *ecuación de Bellman* Para cualquier política π y cualquier estado s, la siguiente condición se mantiene entre el valor de s y el valor de sus posibles sucesores:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) \left[r(s, a, s') + \gamma E_\pi \left[\sum_{k=0}^{\infty} R_{t+k+2} | S_{t+1} = s' \right] \right] \tag{2.3}$$

Resolver una tarea de aprendizaje por refuerzo significa, brevemente, encontrar una

política que consiga mucha recompensa en el largo plazo. Para MDPs finitos, podemos definir una política óptima de la siguiente manera. Una política π es mejor o igual que otra política π' si su recompensa esperada es mayor o igual que la de π' para todos los estados. Siempre hay al menos una política que es mejor o igual que todas las demás. Esta es una política óptima, que aunque pueden ser más de una, las denotamos como π_*.

Hemos definido funciones de valor óptimas y políticas óptimas. Sin embargo, lograr que un agente aprenda estas cosas en la práctica rara vez sucede. Para el tipo de tareas en que estamos interesados, las políticas óptimas sólo pueden ser generadas con un costo computacional extremo. El enfoque que se toma es usar aproximaciones, por ejemplo, hay mucho estados que el agente enfrentará con muy poca probabilidad, así que tomar acciones subóptimas en estos estados no es muy relevante en las recompensas que el agente recibe. La naturaleza online del aprendizaje por refuerzo hace que sea posible aproximar políticas óptimas en formas que ponen más esfuerzo en aprender a tomar buenas decisiones para estados encontrados frecuentemente. Esta es una propiedad clave que distingue el aprendizaje por refuerzo de otros enfoques para resolver aproximadamente MDPs.

Siempre existe el problema de balancear la exploración con la explotación. Siendo que la explotación nos permite utilizar los conocimientos que ya tenemos para obtener la máxima recompensa, pero la exploración nos permite obtener un mejor conocimiento y así mejorar nuestras decisiones. Si sólo se hiciera exploración, no tendría sentido puesto que nunca se explotaría el conocimiento generado. Y si siempre se hiciera explotación, quedaríamos atorados en la toma de decisiones sub-óptimas. Lo ideal es que haya una alta tasa de exploración al comienzo y que esta vaya disminuyendo hasta un valor pequeño estable con el tiempo.

2.1.2. Iteración Generalizada de Política (GPI)

La iteración de política consiste en dos procesos simultáneos que interactúan. Uno hace que la función de valor sea consistente con la política actual (evaluación de política), y el otro hace a la política avara respecto a la función de valor actual (mejora de política). En la iteración de política, estos procesos se alternan, completándose uno antes de que el otro comience, pero esto no es realmente necesario.

Llamamos Generalized Policy Iteration (GPI) a la idea de dejar que ambos procesos, la evaluación de la política y la mejora de la política, interactúen, independientemente de la granularidad y otros detalles de los dos procesos [Van Der Wal, 1978]. El esquema general de GPI esta ilustrado en la figura 2.1.

2.1.3. On-policy vs Off-policy

Los métodos on-policy tratan de evaluar y mejorar la política que está siendo usada para tomar decisiones. Imagine en cambio que queremos mejorar una determinada política π estimando v_π o q_π; pero sólo disponemos de episodios generados por otra política $\mu \neq \pi$.

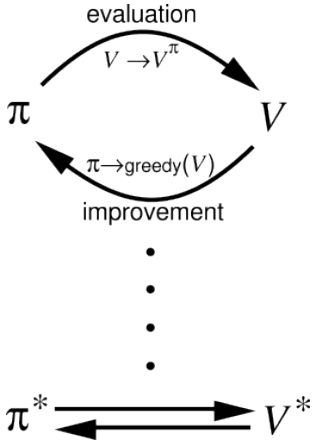


Figura 2.1: GPI

Llamamos a π la política objetivo, porque aprender su función de valor es el objetivo del proceso de aprendizaje, mientras que a μ la llamamos política de comportamiento porque es la política que controla al agente y genera el comportamiento [Precup et al., 2001].

2.1.4. Método actor-critic

Consiste en tener una estructura de memoria separada para representar explícitamente la política independientemente de la función de valor. La estructura de la política es conocida como el actor, porque se usa para seleccionar acciones, y la función estimada de valor es conocida como el crítico, porque critica las acciones hechas por el actor [BARTO, 2004].

Las críticas toman la forma del error, por ejemplo, en el caso de diferencias temporales, la crítica sería:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2.4}$$

Donde V es la función de valor actual implementada por el crítico. Este error puede ser usado para evaluar la acción recién tomada A_t. Si el error es positivo, la tendencia a seleccionar A_t debería incrementar, pero si el error es negativo, dicha tendencia debería disminuir.

2.1.5. Políticas parametrizadas

Cuando el número de estados es demasiado grande, ya no es conveniente mantener promedios separados para cada uno en una tabla. En su lugar, el agente puede mantener la función de valor v_π o q_π como funciones parametrizadas y ajustar dichos parámetros

para ajustarse mejor a los retornos observados. Esto también puede producir estimaciones precisas, aunque depende mucho del aproximador de función parametrizado que se escoja.

2.1.6. Métodos de Gradiente de Política

Son métodos de Reinforcement Learning (RL) que optimizan políticas parametrizadas respecto al retorno esperado usando la gradiente descendente [Sutton et al., 1999].

Se asume que podemos modelar el sistema en forma de tiempo discreto y denotaremos el tiempo presente como t. Para tener en cuenta el factor estocástico del modelo, denotamos el cambio de estados usando una distribución de probabilidad $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ como modelo donde $a_t \in \mathbb{R}^N$ denota la acción actual, y $s_t, s_{t+1} \in \mathbb{R}^N$ denotan los estados actual y siguiente. También se asume que las acciones son generadas por una política $\pi(a_t|s_t)$ que se modela como una función de probabilidad para incorporar acciones exploratorias. Se asume que esta política está parametrizada por k parámetros $\theta \in \mathbb{R}^k$. La secuencia de estados y acciones forman una trayectoria denotada por $\tau = [s_{0:H}, a_{0:H}]$ donde H denota el horizonte que puede ser infinito. En cada instante de tiempo, el sistema recibe una recompensa denotada $r_t = r(s_t, a_t) \in \mathbb{R}$.

El objetivo general de la optimización de política en RL es optimizar los parámetros de la política $\theta \in \mathbb{R}^k$ de forma que el retorno esperado se optimize.

$$J(\theta) = \left\{ \sum_{t=0}^{H} \gamma^t r_t \right\} \tag{2.5}$$

Para eso, los métodos de gradiente de política seguirán la gradiente ascendente del retorno esperado para actualizar los parámetros $\theta \in \mathbb{R}^k$.

$$\theta_{h+1} = \theta_h + \alpha_h \nabla_\theta J_{\theta=\theta_h} \tag{2.6}$$

Donde $\alpha_h \in \mathbb{R}^+$ es la tasa de aprendizaje y $h = \{0, 1, 2, ...\}$ es el número de la actualización actual. Nótese que el tiempo t y el número de actualización h son diferentes, por ejemplo, imagine un aprendizaje episódico donde la actualización se realiza sólo al final de cada episodio.

2.2. Redes Neuronales y Deep Learning

2.2.1. Multilayer perceptron

El Multilayer Perceptron (MLP) es una red neuronal formada por mútliples capas que le permiten aproximar funciones no lineales [Yang, 2010]. La arquitectura del percep-

tron multicapa la podemos observar en la figura 2.2.

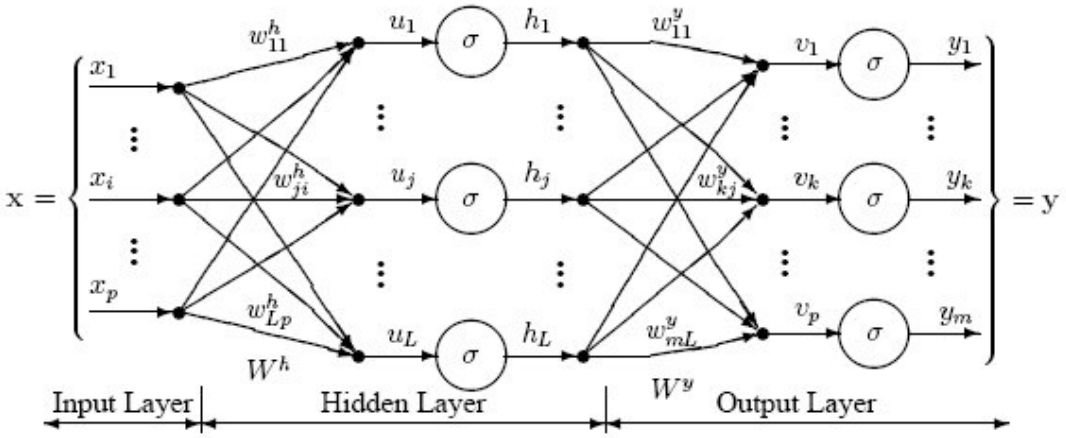


Figura 2.2: MLP

Básicamente tenemos entradas x_i y salidas y_i. La red neuronal es una función paramétrica con parámetros w que aproxima $y = f(x)$ modificando sus parámetros. Podemos ver a la red neuronal como una función $g(x, w) \sim f(x)$. El algoritmo que usa para aproximarse más a $f(x)$ es el de la gradiente descendente del error respecto a sus parámetros. El error puede ser cualquier función de error entre $f(x)$ y $g(x, w)$, siendo la más común la función de error cuadrático.

$$w = w - \alpha \nabla_w E \tag{2.7}$$

2.2.2. Restricted Boltzman Machine

Las Restricted Boltzman Machine (RBM) son redes neuronales generativas estocásticas que pueden aprender una distribución de probabilidad sobre su conjunto de entradas [Hinton, 2010]. La arquitectura de una RBM se encuentra en la figura 2.3. Consiste de nodos de unidades visibles x y unidades escondidas h. Cada unidad visible está conectada a todas las unidades escondidas y viceversa. Cada conexión tiene un peso w asociado, que es el parámetro que se busca optimizar mediante el entrenamiento. Cada unidad visible e invisible está desplazada por un bias b o c. Las unidades visibles representan data observable mientras que las unidades escondidas describen las dependencias entre las variables observadas.

Para hallar las probabilidades se hace uso de una función de energía:

$$E(x, h) = -\sum_j \sum_k w_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \tag{2.8}$$

El entrenamiento de esta red consiste básicamente en maximizar la probabilidad del

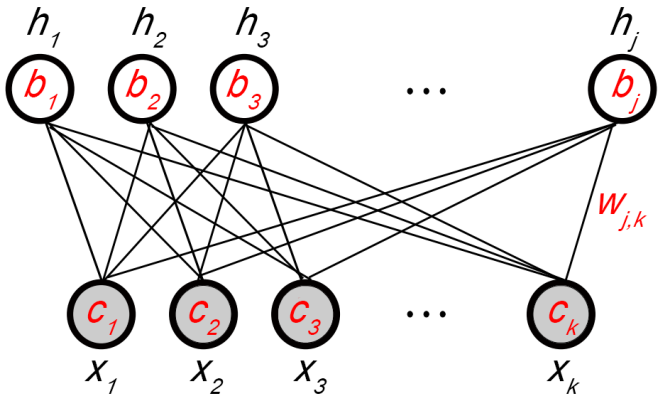


Figura 2.3: Arquitectura de la RBM

valor de entrada. Para hacer esto se minimiza el promedio de la probabilidad logarítmica usando la gradiente descendente.

2.2.3. Deep Belief Network

En [Hinton and Salakhutdinov, 2006] se demostró que las RBM podían ser apiladas y entrenadas de forma voraz para formar las llamadas Deep Belief Networks [Erhan et al., 2010]. Las Deep Belief Network (DBN) son modelos que aprenden a extraer una representación jerárquica profunda de los datos de entrenamiento. Modelan la distribución conjunta de un vector observado x y las l capas ocultas h^k como sigue:

$$P(x, h^1, ..., h^k) = \left(\prod_{k=0}^{l-2} P(h^k | h^{k+1}) \right) P(h^{l-1}, h^l) \tag{2.9}$$

Donde $x = h^0$, $P(h^{k-1}, h^k)$ es una distribución condicional para las unidades visibles condicionadas por las unidades ocultas de la RBM en el nivel k, y $P(h^{l-1}, h^l)$ es la distribución conjunta de las capas visible e invisible en el nivel más alto de la RBM. Esto se ilustra en la figura 2.4

El principio de entrenamiento no supervisado goloso a nivel de capas puede ser aplicado a las DBN con las RBM como el bloque de construcción de cada capa [Hinton and Salakhutdinov, 20 Bengio et al., 2007]. El proceso es como sigue:

1. Entrenar la primera capa como una RBM que modela la entrada $x = h^0$ como su capa visible

2. Usar esta primera capa para obtener una representación de la entrada que será utilizada como data para la segunda capa. Existen dos soluciones comunes, escoger la representación como las activaciones promedio $p(h^1 = 1|h^0)$ o como muestras de $p(h^1|h^0)$

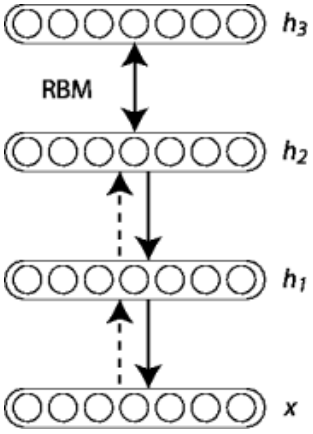


Figura 2.4: Pila de RBM

3. Entrenar la segunda capa como una RBM, tomando la data transformada del paso anterior como ejemplos de entrenamiento para la capa visible de esa RBM

4. Iterar (2 y 3) para el número deseado de capas, propagando haci arriba las muestras o activaciones promedio en cada paso

5. Ajuste fino de todos los parámetros de esta arquitectura profunda usando un criterio de aprendizaje supervisado. Por ejemplo, usando la gradiente descendente.

2.3. Consideraciones Finales

En el siguiente capítulo veremos cómo estas ideas pueden ser usadas juntas para potenciarse mutuamente y crear un algoritmo de aprendizaje por refuerzo multiagente. Y a pesar de que surgen ciertas inconsistencias, estas son superadas con la introducción de nuevas estrategias que veremos a continuación.

Capítulo 3

Estado del Arte

3.1. Agent2D

Agent2D es un equipo de fútbol de muestra para la categoría de simulación 2D de la RoboCup. Forma parte del código base de HELIOS, un equipo que viene participando en competiciones de la RoboCup desde el año 2000. Ha ganado 2 veces el primer puesto y 2 veces el segundo puesto, y se ha mantenido entre los 3 primeros desde el 2007.

La primera vez que se publico el código base de HELIOS fue el 2006. Desde entonces está en constante mantenimiento. Este código es el más popular desde el 2012 y ha sido usado por un amplio número de equipos para participar por primera vez. En [Akiyama and Nakashima, 2013] podemos encontrar información más detallada sobre este programa.

3.2. Deep Q Network

Esta arquitectura aparece por primera vez en [Mnih et al., 2015]. La idea básica es usar una Red Neuronal Profunda para aproximar la función de valor de las acciones. Con esta nueva arquitectura es posible aprender directamente de la información sensorial cruda. En este trabajo se usa una Convolusional Neural Network (CNN) para leer directamente los píxeles de la pantalla de 49 juegos de atari. Sólo con esta información y el score se logra que el agente aprenda a jugar con desempeño sobrehumano en más de la mitad de los juegos.

Como usar aproximadores no lineales en RL ha probado ser inestable, se aplican dos ideas clave:

- Usar un mecanismo biológicamente inspirado llamado replay de experiencias
- Los valores objetivos son actualizados periodicamente, reduciendo así correlaciones

A pesar de que con DQN se pueden resolver problemas con estados de alta dimensionalidad; sólo puedo manejar acciones discretas y de baja dimensionalidad. Una solución obvia para adaptar DQN a espacios de acciones continuas sería discretizar las acciones. Sin embargo esto tiene muchas limitaciones, especialmente la maldición de la dimensionalidad. Por ejemplo, en un sistema con 7 grados de libertad (como el brazo humano) con la discretización más gruesa $a_i \in -k, 0, k$, obtenemos un espacion de acciones con dimensionalidad $3^7 = 2187$.

3.3. Deep Reinforcement Learning con espacio de acciones continuas

En [Silver et al., 2014] se propone un algoritmo eficiente de RL que puede ser aplicado con un espacio de acciones continuas. Este algoritmo usa la gradiente de una política determinista en un modelo actor-critic que permite usar otra política estocástica para lograr una exploración adecuada.

En [Lillicrap et al., 2015] se hace lo mismo que en el trabajo anterior, pero mejora la estabilida del sistema utilizando los métodos de DQN. Propone un algoritmo actor-critic libre de modelo, off-policy que utiliza aproximadores de funciones profundos para aprender políticas en espacios de acción continuos de varias dimensiones.

Capítulo 4

Propuesta

Aplicar Deep Q Learning al aprendizaje de jugadas de futbol de bajo nivel.

- Se tiene un simulador hecho a la medida
- Se necesita un sistema de scoring que le de recompensas al agente por pasos intermedios, como coger el balon, patearlo, perderlo, sacar el balon fuera del campo y hacer un gol.
- Es necesario un aprendizaje icremental, donde la primera etapa, la jugaria con un adversario que no se mueve, luego con un adversario aleatorio y finalmente con una estrategia de arbol de decision
- Tambien podemos escalar este aprendizaje en un 1 vs 1, 2 vs 2 y finalmente un 3 vs 3
- Para tener una idea de que tan bien se desempeña el algoritmo, lo compararemos con el desempeño de un jugador humano
- Se utilizaran 2 modelos de aprendizaje, el primero un deep q learning con una mlp por detras, y el segundo un deep q learning con una red convolucional por detras, que aprendera directamente de los frames generados por el juego

Capítulo 5

Pruebas y Resultados

A continuación se describirán las pruebas y resultados de trabajos similares, además de mostrar la planificación de pruebas para el presente trabajo.

5.1. Pruebas y resultados en trabajos similares

En [Mnih et al., 2015] se usan como métricas el score promedio por episodio y el valor de acción promedio (Figura 5.1). Luego se compara la eficiencia del agente de RL con un jugador humano experto, siendo que el jugador humano tiene un 100 % de eficiencia, el agente de DQN logra más de 100 % de eficiencia en 29 juegos de un total de 49 (Figura 5.2).

En [Silver et al., 2014] se usa una versión continua del problema de los bandidos para hacer una comparación directa entre las gradientes de política estocástica y deterministica. Luego se usan problemas donde el espacio de acciones es continuo y se comparan varios algoritmos recientes de RL usando como métrica la recompensa total por episodio (Figura 5.3).

En [Lillicrap et al., 2015] se utilizaron ambientes físicos simulados de varios niveles de dificultad para probar el algoritmo. Esto incluyo ambientes clásicos de RL como cartpole, así como también tareas difíciles de alta dimensionalidad como brazos manipuladores o locomoción de esqueletos. Para la comparación se usa la recompensa normalizada (Figura 5.4).

5.2. Planificación de pruebas

Durante el entrenamiento del equipo se pretende registrar la recompensa por episodio, para evaluar la velocidad de convergencia.

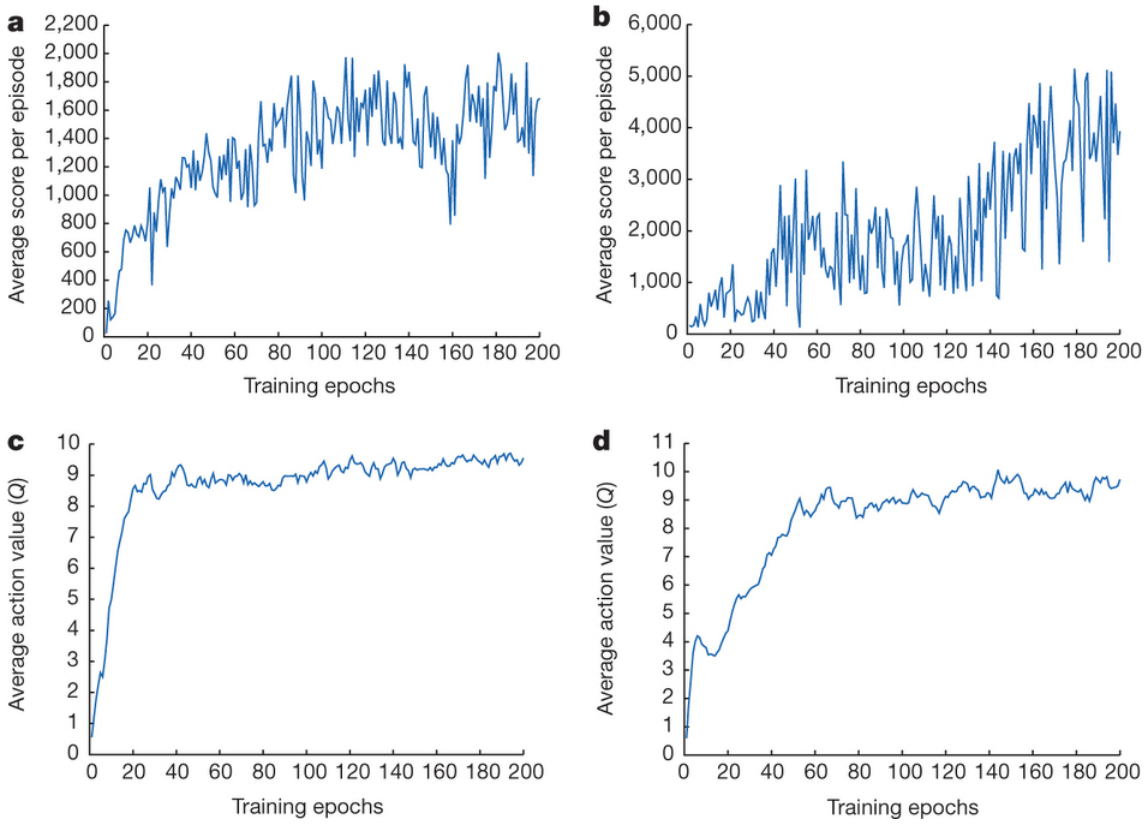


Figura 5.1: Resultados con DQN

Una vez que el equipo esté listo, se jugarán 50 partidos contra agent2D, registrando partidos ganados, perdidos y cantidad de goles. Si el número de partidos ganados es superior al 60 %, diremos que alcanzamos el objetivo.

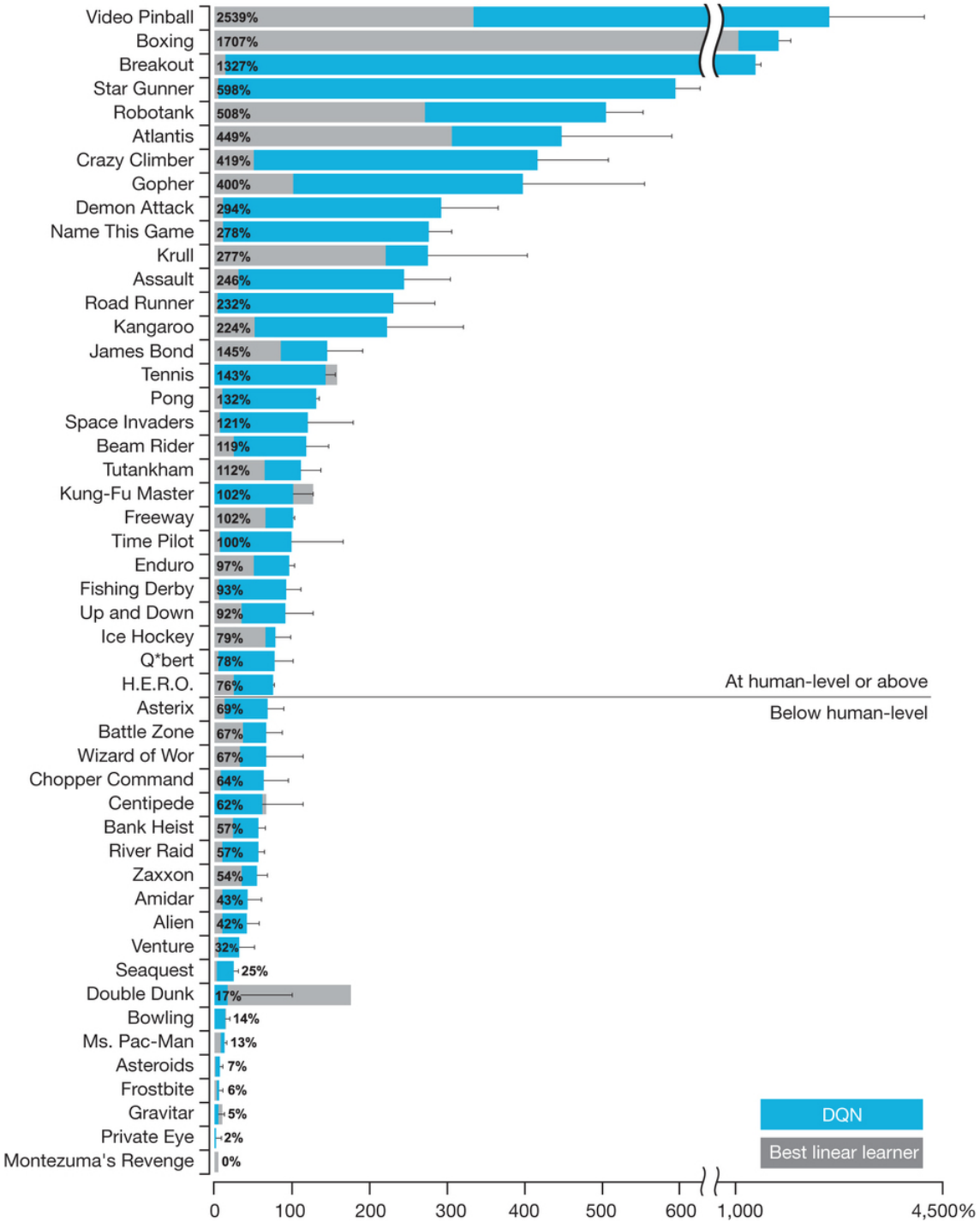
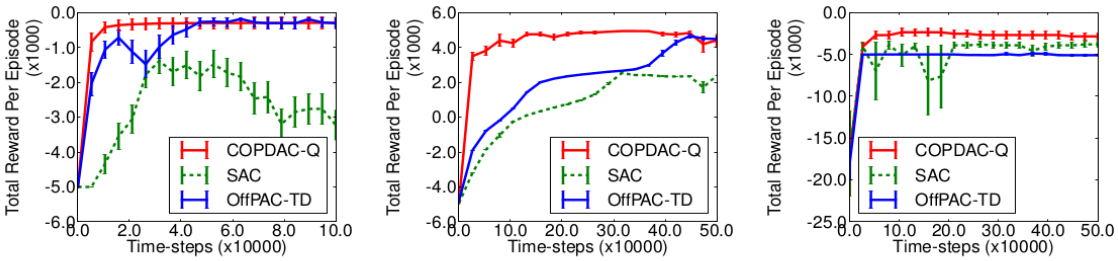


Figura 5.2: DQN vs humano experto



(a) Mountain Car (b) Pendulum (c) 2D Puddle World

Figura 5.3: Resultados de DPG

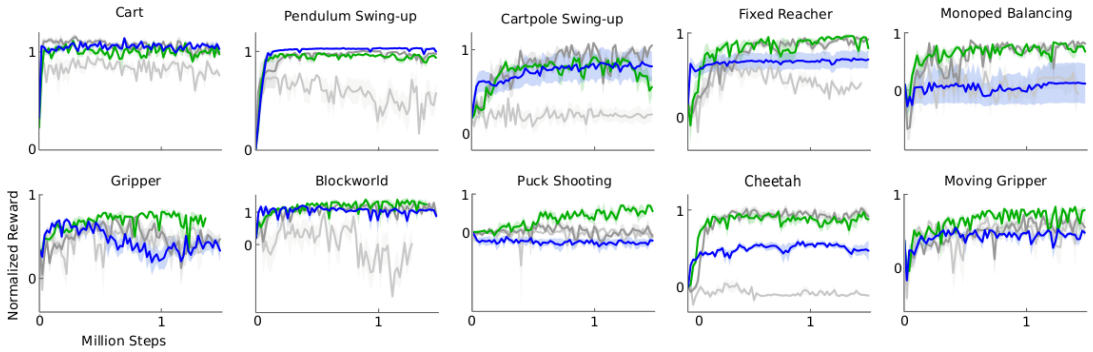


Figura 5.4: Resultados de DPG modificado

Capítulo 6

Conclusiones y Trabajos Futuros

Se vio que los algoritmos de aprendizaje por refuerzo han recibido un impulso con la aparición de la Deep Q-Network, ya que nuevamente han llamado la atención de los investigadores y se están desarrollando varias líneas de investigación relacionadas con el tema. Una de esas líneas es la aplicación del aprendizaje por refuerzo en contextos con espacios de acciones continuas.

Con Deep Q-Network es posible calcular el valor de las acciones con una función paramétrica. Siendo las más comúnmente utilizadas las redes neuronales profundas, ya que permiten trabajar con grandes cantidades de estados continuos. Sin embargo, su aplicación no es trivial, ya que se generan inestailidades que se resuelven con la aplicación de dos ideas claves. La primera es usar una memoria de replay, y la segunda, actualizar los valores de las acciones periódicamente.

6.1. Problemas encontrados

Falta una mejor aplicación de las estrategias multiagente. Por ahora sólo se tiene un aprendizaje independiente.

6.2. Recomendaciones

Se recomienda utilizar un código base de alguno de los equipos ganadores de la categoría de simulación de la RoboCup de años pasados. Ya que la programación de un equipo básico, capaz de comunicarse y sincronizarse con el servidor, además de modelar el mundo, no es sencilla.

6.3. Trabajos futuros

Como trabajo futuro sería bueno implementar mejores estrategias de aprendizaje multiagente, como por ejemplo aprendizaje con valores de influencia.

Bibliografía

[Akiyama and Nakashima, 2013] Akiyama, H. and Nakashima, T. (2013). Helios base: An open source package for the robocup soccer 2d simulation. In *RoboCup 2013: Robot World Cup XVII*, pages 528–535. Springer.

[BARTO, 2004] BARTO, M. (2004). J. 4 supervised actor-critic reinforcement learning. *Handbook of learning and approximate dynamic programming*, 2:359.

[Bengio et al., 2007] Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153.

[Claus and Boutilier, 1998] Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, pages 746–752.

[Erhan et al., 2010] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660.

[Hinton, 2010] Hinton, G. (2010). A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Precup et al., 2001] Precup, D., Sutton, R. S., and Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. In *ICML*, pages 417–424.

[Silver et al., 2014] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *ICML*.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press.

[Sutton et al., 1999] Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999). Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063.

[Van Der Wal, 1978] Van Der Wal, J. (1978). Discounted markov games: generalized policy iteration method. *Journal of Optimization Theory and Applications*, 25(1):125–138.

[Yang, 2010] Yang, Z. R. (2010). Multi-layer perceptron. In *Machine Learning Approaches To Bioinformatics*, pages 133–153. World Scientific.