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
Mayo 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.

El presente trabajo aplica este mismo algoritmo llamado DQN a la creación de un equipo de fútbol que juegue en la categoría de Robocup simulación de fútbol 2D. Se añade además la complejidad de cómo hacer que varias de estas redes interactúen en un sistema multiagente como es un equipo de fútbol.

1.2. Planteamiento del Problema

El fútbol de robots simulado es un problema que se puede modelar con un Markov Decision Process (MDP) con número de estados y acciones virtualmente infinitos. El problema es implementar un algoritmo que sea capaz de tomar decisiones en este contexto y jugar de la mejor manera posible, teniendo en cuenta que cada jugador es un agente independiente y con capacidad de comunicación limitada.

1.3. Objetivos

Aplicar el algoritmo DQN en la construcción de un equipo multiagente de fútbol simulado 2D que juegue mejor que un equipo estándar agent2d.

1.3.1. Objetivos Específicos

- Modelar el problema como un MDP con sus respectivos estados, acciones y función de recompensa
- Implementar el algoritmo DQN en cada agente del equipo y entrenarlo
- Analizar el desempeño del equipo contra un equipo estándar agent2d

1.4. Organización de la tesis

En el capítulo 2 se describirá el problema de forma un poco detallada, el funcionamiento del servidor de fútbol simulado, los modelos de comunicación, percepción y acción de los agentes en el campo de juego.

En el capítulo 3 tenemos el marco teórico, en el cual se hablará de tres temas: el aprendizaje por refuerzo, los sistemas multiagente y el aprendizaje profundo. En cada uno de ellos se tocarán conceptos necesarios para el desarrollo de la propuesta.

En el capítulo 4 hablaremos del estado del arte. Veremos el equipo de fútbol simulado estándar agent2d. Repasaremos conceptos de Deep Q-Learning, una técnica muy reciente que combina apredizaje por refuerzo y aprendizaje profundo, por lo que algunos la llaman Deep Reinforcement Learning. Y finalmente hablaremos del aprendizaje por refuerzo con espacios de acciones continuas.

En el capítulo 5 se presentará la propuesta, que consiste básicamente en cómo se va a utilizar el algoritmo de Deep Q-Learning para jugar al fútbol en un entorno simulado.

Finalmente en el capítulo 6, mostraremos los resultados de trabajos similares y una planificación de las pruebas que se realizarán al equipo una vez que esté terminado.

Capítulo 2

El problema

En este capítulo se detallará a grandes rasgos cómo funciona el simulador de fútbol y los problemas que hay que tener en cuenta a la hora de diseñar un equipo.

2.1. Servidor

Es un sistema que permite a dos equipos jugar un partido de fútbol. La comunicación entre el servidor y los clientes se realiza mediante UDP/IP. Cada cliente es un proceso separado y se comunica con el servidor mediante un puerto. Cada equipo puede tener hasta 12 clientes (10 jugadores, 1 portero y 1 coach).

Los jugadores envían peticiones al servidor indicando las acciones que quieren realizar (patear, girar, correr, etc) y el servidor recibe estos mensajes y actualiza el ambiente acorde a las acciones de todos los clientes. Adicionalmente, el servidor provee a los clientes de información sensorial.

Es importante recalcar que el servidor es un sistema en tiempo real que trabaja con intervalos de tiempo discretos (o ciclos). Cada ciclo tiene una duración específica, así que las acciones que necesitan ser ejecutadas en un determinado ciclo deben llegar al servidor en el intervalo de tiempo correcto. Así, un bajo desempeño en un jugador puede resultar en oportunidades de acción perdidas que al final pueden perjudicar al equipo en general.

2.1.1. Reglas de juego

Kick-off Se da inicio al juego, puede ser al inicio del primer o segundo tiempo o después de un gol. Los jugadores deben estar en su lado del campo para que esto pueda ocurrir. Se da un tiempo para que los jugadores se teletransporten a su posición deseada.

Gol Se anuncia a todos los jugadores que hubo un gol y se pasa al estado Kick-off.

Fuera de juego Cuando el balón sale del campo de juego, el referee mueve el balón a la posición apropiada, una línea lateral, una esquina o el área de gol, dependiendo de por donde salga el balón.

Offside Un jugador está en offside si:

- Está en la mitad rival del campo
- Está más cerca al arco rival que al menos 2 defensas
- Está más cerca al arco que la pelota
- Está más cerca al balón que 2,5 metros

Backpass Al igual que en los partidos de fútbol reales, el arquero no puede atrapar el balón con las manos si esté le ha sido pasado por un jugador de su propio equipo. Esto terminaría en un tiro libre.

Faltas en tiro libre Al hacer un tiro libre, un jugador no puede pasarse el balón a si mismo. Esto resultaría en un tiro libre para el otro equipo.

Muerte súbita Si al terminar los 2 tiempos hay un empate, el partido continúa hasta que uno de los dos equipos anote un gol, el equipo que lo logre será el ganador.

2.1.2. Mensajes al cliente

El servidor puede enviarle los siguientes mensajes a los clientes: Escuchar mensaje de otro jugador, Mirar y Sentir cuerpo.

2.2. Cliente

Hay un cliente independiente por cada jugador, que le envía peticiones al servidor para realizar acciones y para consultar el estado actual. Puede enviarle los siguientes mensajes al servidor: Atrapar, Cambiar vista, Correr, Patear, Teletransportarse, Decir mensaje, Sentir cuerpo, Pedir marcador, Girar y Girar cuello. Ya que la comunicación con el servidor es vía UDP/IP, el cliente puede estar implementado en cualquier lenguaje de programación.

2.2.1. Mensajes entre los jugadores

Cada jugador puede escuchar a lo mucho un mensaje de cada equipo por cada ciclo de simulación. Un jugador sólo puede oír los mensajes de jugadores cercanos (dentro de

un radio definido en el servidor). Sin embargo todos los jugadores pueden escuchar los mensajes del referee.

Si llegan más mensajes de los que un jugador puede escuchar, se escoge uno aleatoriamente y el resto se descarta. Es importante notar que un jugador puede enviar y recibir un mensaje al mismo tiempo.

2.2.2. Percepción del ambiente

Los agentes perciben el ambiente mediante el sensor de visión, que contrario a lo que se podría pensar, no retorna imágenes, sino la información ya procesada de lo que hay en el campo de visión. Es decir, el servidor envía a los clientes una lista de los objetos que el cliente ve en el campo de juego, junto con su dirección y su distancia. Entre estos objetos pueden aparecer jugadores del mismo equipo o rivales, las porterías, el balón o banderines que son usados para la ubicación de los jugadores.

Es importante notar que los agentes no conocen su posición (x, y) dentro del juego, sino que tienen que deducirla a partir de los banderines que hay en el campo tal como muestra la figura 2.1.

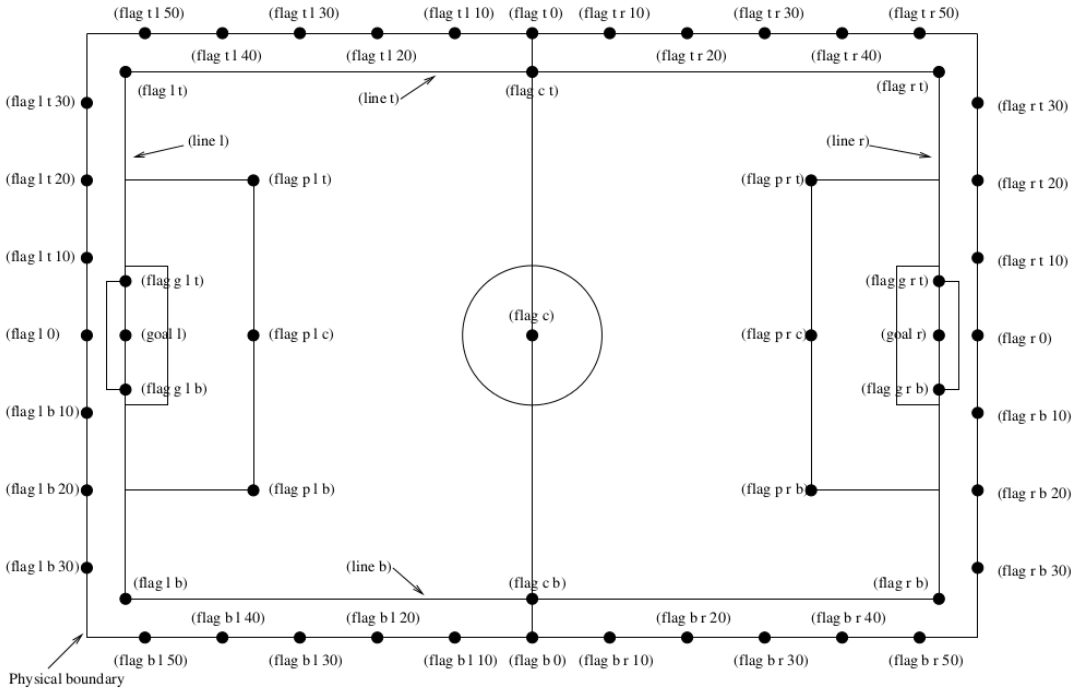


Figura 2.1: Banderas en el campo de juego

Además del sensor de visión, tenemos otro sensor de cuerpo que reporta el estado físico del jugador. Por ejemplo la estamina (necesaria para correr), la velocidad, el ángulo de la cabeza.

2.3. Consideraciones finales

En resumen, los problemas son que tenemos 11 agentes que deben jugar juntos con una capacidad limitada de comunicación. Que la representación del mundo, a pesar de ser de alto nivel, es muy compleja. Las acciones son continuas y además hay que controlar un gran número de parámetros que afectan cómo los agentes perciben el mundo. En los siguientes capítulos veremos conceptos que poco a poco nos llevarán a la resolución de este problema en la propuesta.

Capítulo 3

Marco teórico

En esta sección se describirán varios conceptos que serán útiles en el desarrollo de las siguientes secciones.

3.1. Reinforcement Learning

Es un área del aprendizaje automático, que se inspira en la psicología conductista. Básicamente consiste en qué acción debe seleccionar un agente dentro de un entorno para maximizar una recompensa en el largo plazo. El medio ambiente es normalmente modelado como un MDP. El planteamiento de un problema en Reinforcement Learning (RL) siempre consta de 3 partes: las sensaciones (que determinan los estados), las acciones, y los objetivos (que determinan las recompensas) [Sutton and Barto, 1998].

Los elementos de un RL son:

- **Estados**, conjunto de características que indican como está el ambiente en cada momento
- **Acciones**, acciones disponibles en cada estado que pueden modificar el ambiente
- **Política**, mapea estados a acciones
- **Función de recompensa**, define el objetivo. Mapea un estado o par de (estado, accion) a una recompensa
- **Función de valor**, es un valor numérico que define que tan bueno es un estado o par de (estado, accion) a largo plazo

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.

Los métodos tradicionales de RL han utilizado tablas como función de valor para mantener el retorno estimado de cada estado. Ejemplos de tales técnicas son los métodos de Monte Carlo, diferencias temporales y Q-learning. Sin embargo cuando el espacio de estados es demasiado grande, estas técnicas son imposibles de aplicar y se necesita el uso de nuevos métodos que serán descritos en este trabajo.

3.1.1. Retorno esperado

El retorno esperado G_t es la cantidad de recompensa futura que se puede esperar partiendo de un estado, como se puede ver en su fórmula en la ecuación (3.1), donde R_t es la recompensa en el tiempo t y T es el último paso del episodio.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T \tag{3.1}$$

Para tareas continuas, no existe un último episodio T, por lo tanto necesitamos un concepto adicional, el descuento γ, tal que $0 \leq \gamma < 1$. En la ecuación (3.2) se aprecia como con el valor de descuento se consigue dar mayor peso a las recompensas más cercanas en el tiempo que aquellas que se encuentran distantes.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{3.2}$$

3.1.2. Generalized Policy Iteration

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 ávara 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 3.1.

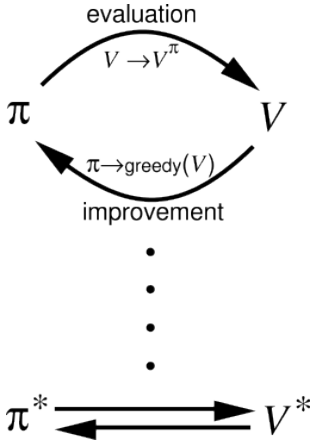


Figura 3.1: GPI

3.1.3. 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 $\mu \neq \pi$. 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].

3.1.4. 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:

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{3.3}$$

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.

3.1.5. 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.

3.1.6. Métodos de Gradiente de Política

Son métodos de 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.

$$J(\theta) = \left\{ \sum_{t=0}^{H} \gamma^t r_t \right\} \tag{3.4}$$

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

$$\theta_{h+1} = \theta_h + \alpha_h \nabla_\theta J_{\theta=\theta_h} \tag{3.5}$$

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.

3.2. Sistemas multiagente

Un sistema multiagente es un sistema compuesto por múltiples agentes inteligentes que interactúan entre ellos. El bloque fundamental de construcción de los sistemas multiagente son los agentes, que aunque no tienen una definición formal y precisa, normalmente son vistos como entidades inteligentes, equivalentes en términos computacionales a un proceso del sistema operativo, que se pueden comunicar mediante un sistema de red.

Los agentes en un sistema multiagente tienen varias características importantes: La autonomía, visión local y descentralización, ya que no hay un agente de control designado.

3.2.1. Aprendizaje en sistemas multiagente

Sólo estudiaremos los modelos de aprendizaje con recompensa, ya que pueden ser modelados como un problema de RL. Entre estos modelos de aprendizaje hay 4 tipos [Claus and Boutilier, 1998]:

- **Aprendizaje como un equipo**, donde los estados y las acciones incluyen a todos los agentes. De hecho se modela a todos los agentes como si fueran uno solo. Requiere de un programa centralizado que vea a todos los agentes como un todo y les indique qué hacer.
- **Aprendizaje independiente**, donde todos los agentes conocen la ubicación de todos los agentes pero las acciones son individuales e independientes. Puede generar competición entre los agentes, llegando a un aprendizaje sub-óptimo.
- **Aprendizaje de acciones conjuntas**, similar al aprendizaje independiente, pero involucra comunicación entre los agentes.
- **Aprendizaje con valores de influencia**, los agentes pueden darse recompensas mutuamente para influenciar las acciones de los otros.

3.3. Redes Neuronales y Deep Learning

3.3.1. 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 perceptron multicapa la podemos observar en la figura 3.2.

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

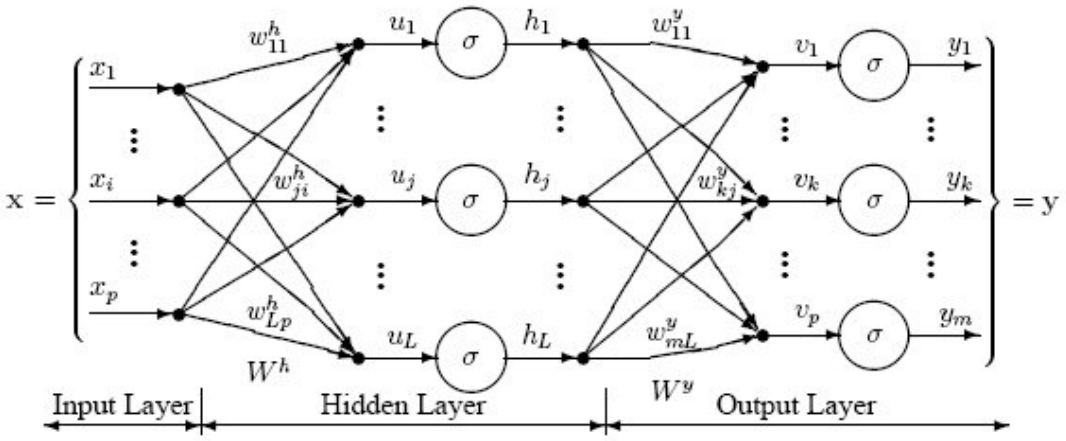


Figura 3.2: MLP

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.

$$w = w - \alpha \nabla_w E \tag{3.6}$$

3.3.2. 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 3.3. 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:

$$E(x, h) = -\sum_j \sum_k w_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \tag{3.7}$$

El entrenamiento de esta red consiste básicamente en maximizar la probabilidad del valor de entrada. Para hacer esto se minimiza el promedio de la probabilidad logarítmica usando la gradiente descendente.

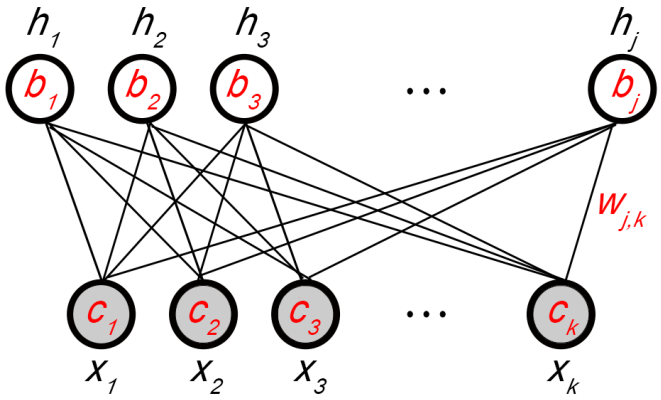


Figura 3.3: Arquitectura de la RBM

3.3.3. 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:

$$P(x, h^1, ..., h^k) = \left(\prod_{k=0}^{l-2} P(h^k | h^{k+1}) \right) P(h^{l-1}, h^l) \tag{3.8}$$

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 3.4

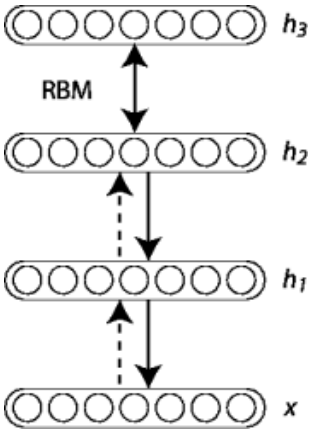


Figura 3.4: Pila de RBM

El principio de entrenamiento no supervisado goloso a nivel de capas puede ser apli-

cado 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 $p(h^1|h^0)$
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.

3.4. 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 4

Estado del Arte

4.1. 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.

4.2. 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 $3^7 = 2187$.

4.3. 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 5

Propuesta

Se propone la creación de un equipo de fútbol para la categoría de simulación 2D de la RoboCup. Este equipo se implementará usando el algoritmo de aprendizaje por refuerzo DQN y se espera que su desempeño sea por lo menos mejor que el equipo de muestra Agent2D que se explicó en la sección 4.1. La definición de los estados y acciones, así como la función de recompensa se encuentran en la siguiente sección.

5.1. Definición de Estados, Acciones y Recompensas

Los estados tendrán la siguiente información:

- $X_{jugador} \in [-54; 54]$
- $Y_{jugador} \in [-32; 32]$
- $dist_{balon} \in \mathbb{R}$
- $dir_{balon} \in [-180; 180]$
- Estamina $\in [0; 4000]$
- Esfuerzo $\in [0; 1]$
- Monto de velocidad $\in \mathbb{R}^+$
- Dirección de la velocidad $\in [-180; 180]$

La información de los jugadores al rededor del agente se proporcionará haciendo un particionamiento de los 360º al rededor del agente en 10 porciones de 36º. A cada porción de 36º le corresponde una sección del estado:

- Equipo $\in \{0, 1, 2\}$
- Distancia $\in \mathbb{R}^+$
- Dirección del cuerpo $\in [-180; 180]$
- Dirección de la cabeza $\in [-90; 90]$
- Visto por última vez $\in [0; 30]$

Se tienen 10 de estos bloques en un estado, uno por cada sección de 36º al rededor del jugador. Con esto, cada estado esta formado por 58 variables continuas, con lo cual queda justificado el uso de redes neuronales profundas.

Para las acciones también se tienen variables continuas, sin embargo se discretizarán para que puedan ser utilizadas con DQN. Las posibles acciones son 34 en total:

- Girar $\in \{-50, -20, -10, 10, 20, 50\}$
- Acelerar $\in \{10, 20, 50, 100\}$
- Patear $\in \{10, 20, 50, 100\} \times \{-50, -20, -10, 10, 20, 50\}$

También se necesita una función de recompensas, que en nuestro caso, funcionará en los siguientes casos:

- Gol a favor: +1
- Gol en contra: -1

5.2. Parámetros de la DQN

Para modelar la función de valor se utilizará una DBN cuya entrada serán los estados definidos en la sección anterior. La salida de esta red neuronal profunda será una neurona de salida por cada una de las 34 acciones. En cada neurona de salida se tendrá el valor de la acción que corresponde a esa neurona. Se tendrá 2 capas ocultas con 50 neuronas en cada una. Además hay varios parámetros que aún falta definir, pero que se definirán en tiempo de implementación, como el tamaño de la memoria de replay, el factor de descuento, el tamaño de los batches, etc.

Capítulo 6

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.

6.1. 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 6.1). 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 6.2).

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 6.3).

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 6.4).

6.2. Planificación de pruebas

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

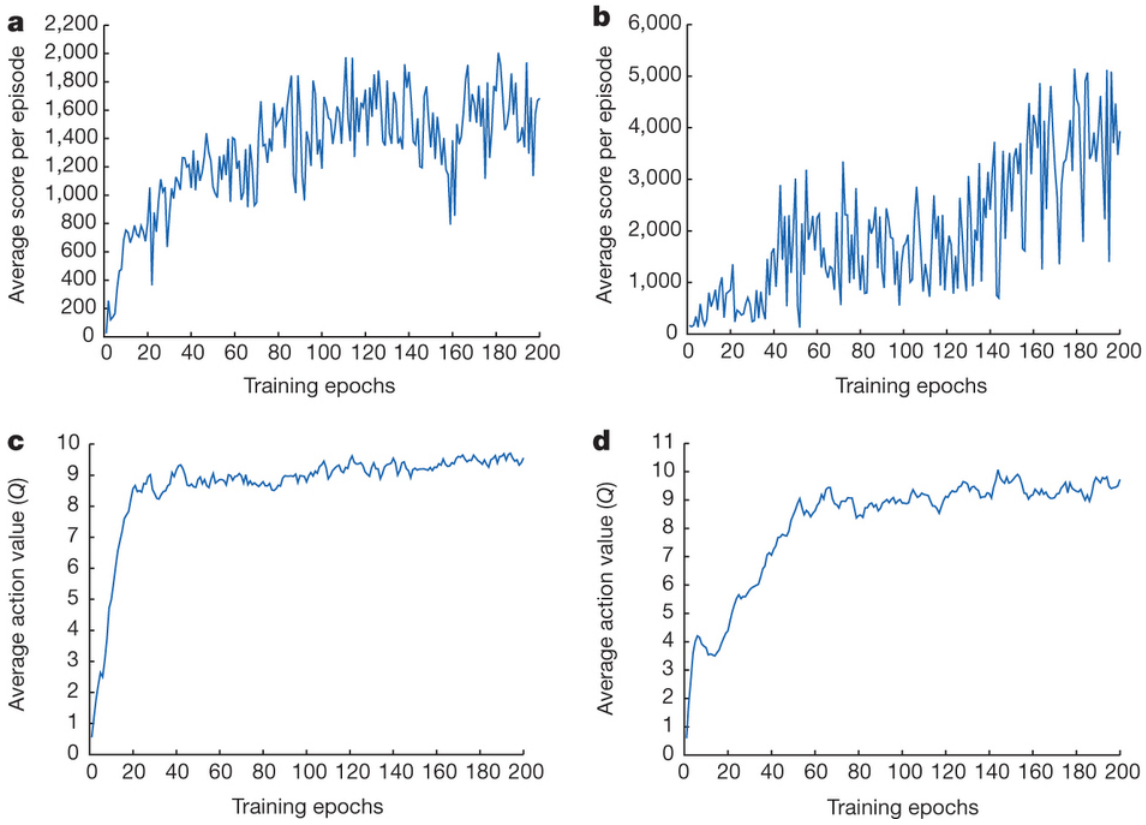


Figura 6.1: Resultados con DQN

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.

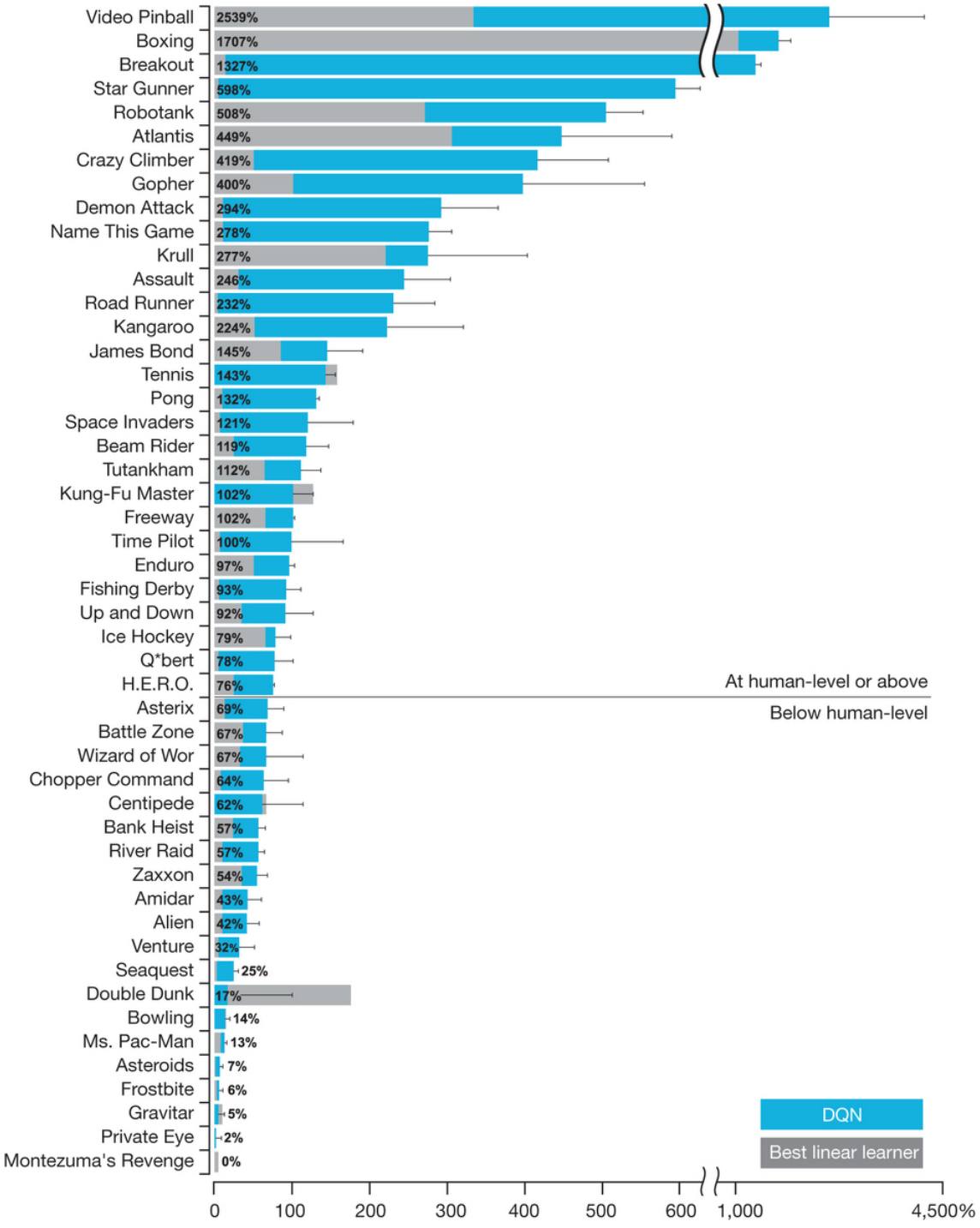
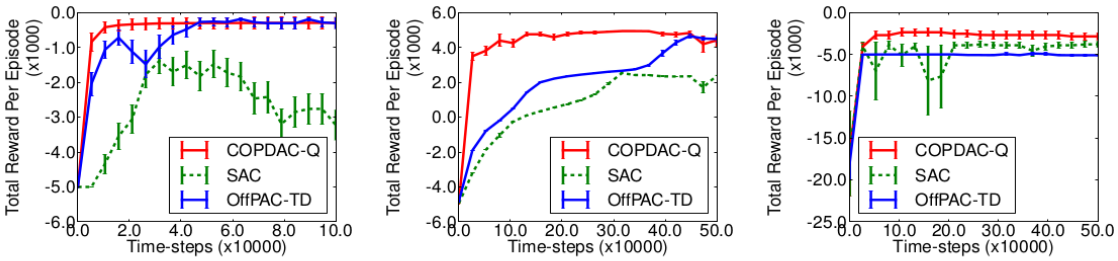


Figura 6.2: DQN vs humano experto



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

Figura 6.3: Resultados de DPG

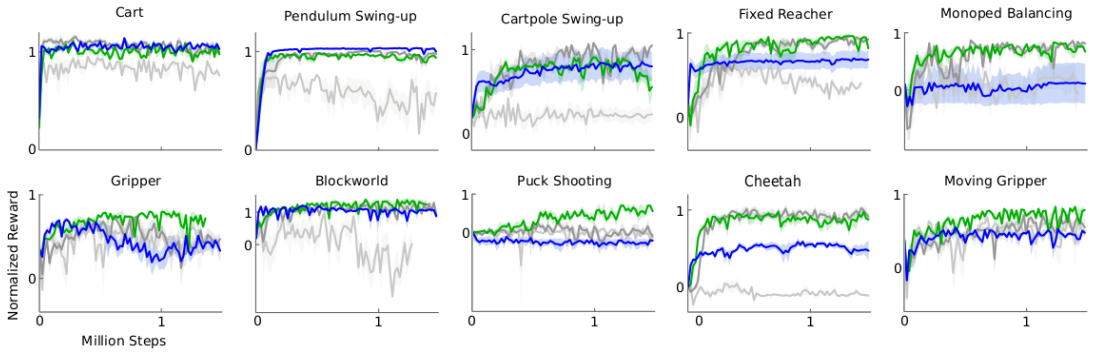


Figura 6.4: Resultados de DPG modificado

Capítulo 7

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.

7.1. Problemas encontrados

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

7.2. 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.

7.3. 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.