# Aprendizaje por Refuerzos

Diplomatura en Ciencia de Datos, Aprendizaje Automático y sus Aplicaciones - FaMAF, 2022

# Agenda

- Problema de los enfoques tabulares.
- Repaso redes neuronales.
- Algoritmos de deep RL.
- Ejemplos de aplicaciones de RL.
- Conclusiones.

# Problema de los enfoques tabulares

# Problema de los enfoques tabulares

¿Qué sucede cuando queremos usar los métodos vistos en entornos más complejos?

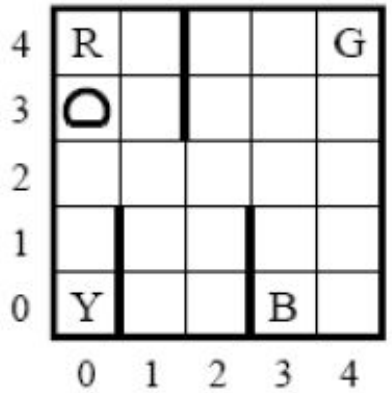
Estados más complejos, pueden por ejemplo involucrar:

- Aumento de la cantidad de estados
- Pasar de ser $s \in \mathbb{N}$ a $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)$
- Pasar a ser continuos, es decir que pueden ser $s \in \mathbb{R}$ o $s_i \in \mathbb{R}$

Acciones más complejas, por ejemplo:

- Aumento de la cantidad de acciones (pudiendo ser continuas)

# Ejemplo 1: Problema del taxi



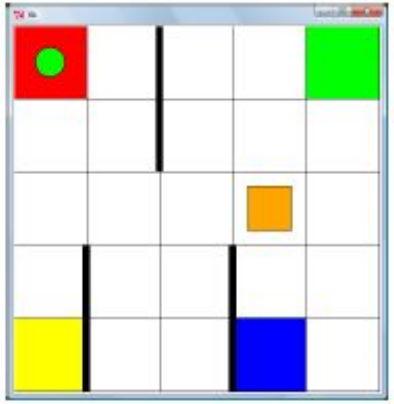



Figure 1: The Taxi Domain.

*25* (ubicaciones taxi) x 5 (ubicaciones pasajero + “en taxi”) x *4* (destinos)

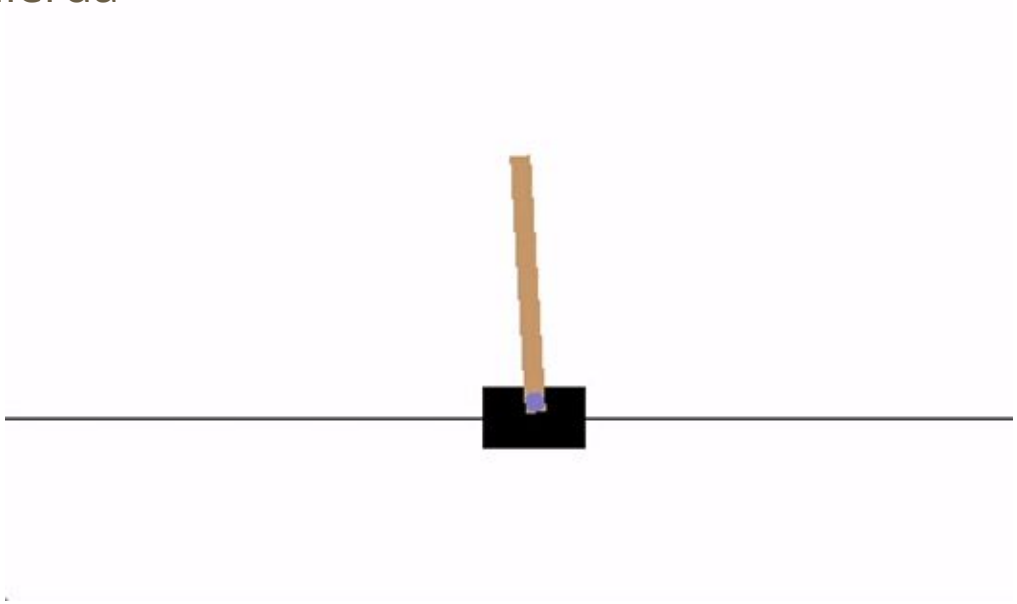
**=> 500 estados**

# Ejemplo 2: Cartpole

Acciones: derecha e izquierda

Estado:

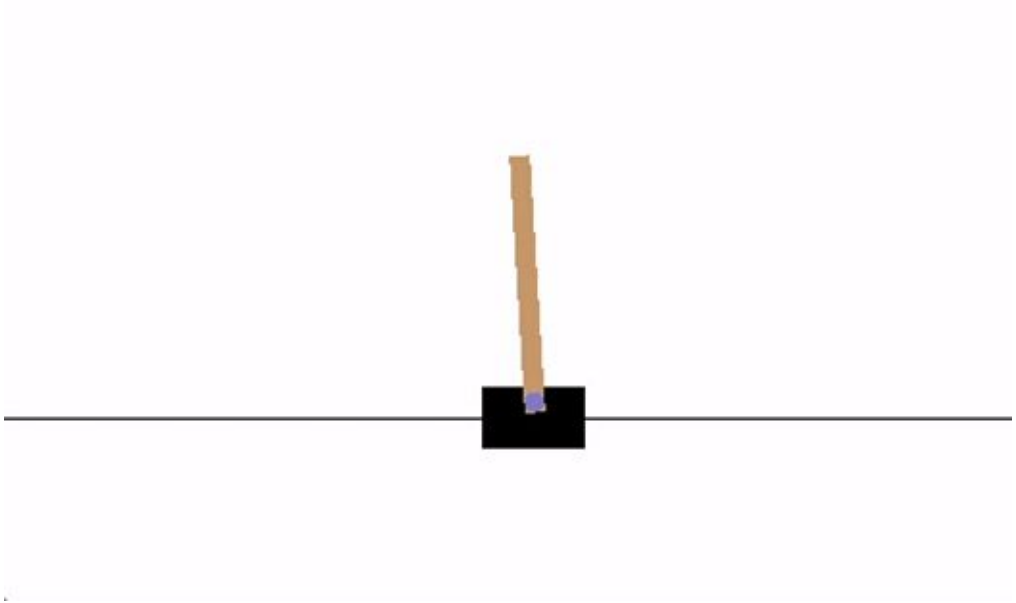
- Posición carrito
- Velocidad carrito
- Ángulo poste
- Velocidad angular



# Posible mitigación: simplificar el problema

Ejemplo: posición del carro

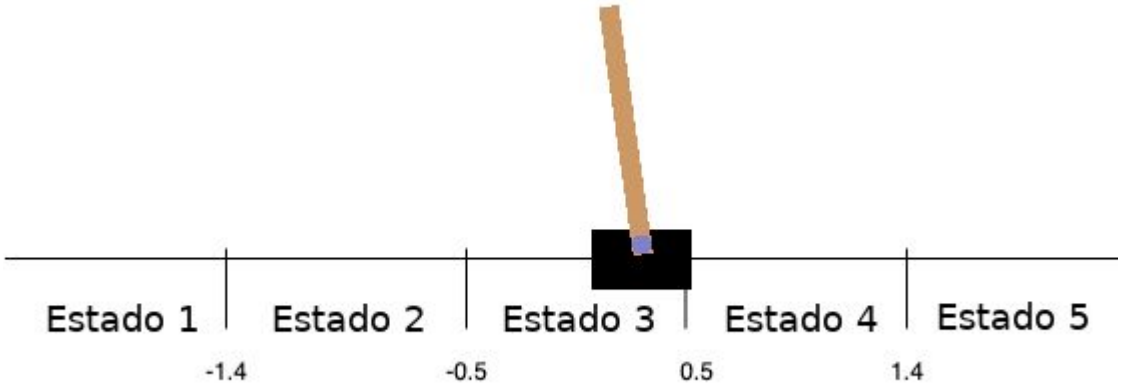
La posición $x$ del carrito es continua, $x \in [-2.5, 2.5]$



# Posible mitigación: simplificar el problema

Transformamos $x \in [-2.5, 2.5]$ en $x \in \{1, 2, 3, 4, 5\}$ (es decir, discretizamos)

Podemos hacer algo similar para los demás estados



# Problemas de simplificar

Simplificar el problema tiene varios inconvenientes. En particular, con la discretización...

- El estado 2  ahora representa todos los valores de -1.4 a -0.5. ¿Y si hay mucha diferencia entre ambos?
- ¿Con qué criterio hacemos la discretización?
- Si se vuelve muy complejo, estaremos manipulando muy artesanalmente los features.

# Solución: aproximar

- Por ejemplo, en lugar de computar la media empírica de la función $Q(s,a)$ para cada $s, a$, obtenerla a partir de un predictor.
- Es decir, usar $\hat{Q}(s,a \mid \theta) \approx Q(s,a)$, donde $\theta$ son los parámetros.
- Los parámetros ahora afectan el valor de **muchos** estados en lugar de sólo uno.

# Solución: aproximar

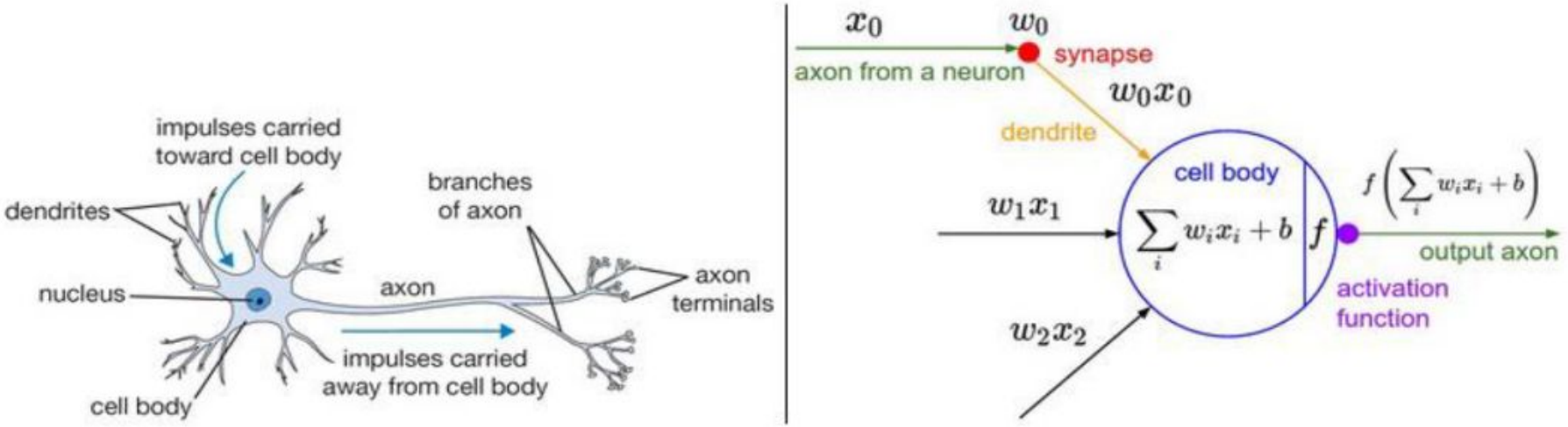
- En la aproximación es donde se juntan principalmente RL con aprendizaje automático.
- Se usan mayoritariamente las redes neuronales (visualización 1, 2).
- La incorporación de ellas + RL forman el **deep RL**.

# Repaso: redes neuronales

Vemos un repaso muy rápido de redes neuronales, que sirven de base de los métodos de aprendizaje por refuerzos profundo

# Repaso redes neuronales

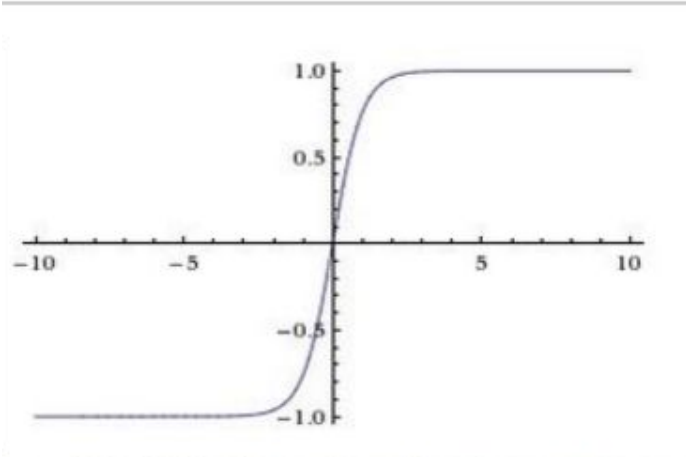
Neurona




A cartoon drawing of a biological neuron (left) and its mathematical model (right).

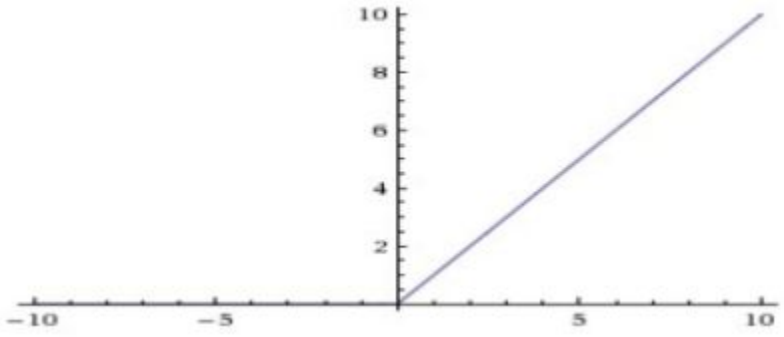
# Repaso redes neuronales

Función de activación. Función sigmoide



# Repaso redes neuronales

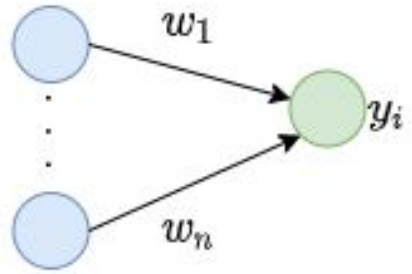
Función de activación. Función ReLU



# Repaso redes neuronales

Conjunto de neuronas -> “redes neuronales”

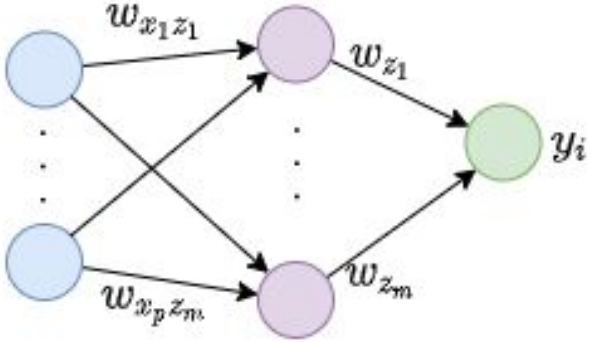
Red neuronal sin capa oculta (regresión lineal)




Capa de entrada $x$ ($P$ neuronas)

Capa de salida $y$

Red neuronal de ejemplo con 1 capa oculta




Capa de entrada $x$ ($P$ neuronas)

Capa oculta $z$ ($M$ neuronas)

Capa de salida $y$

# Repaso redes neuronales

Descenso de gradiente: actualizar los pesos en proporción al gradiente de una función, por ejemplo mediante

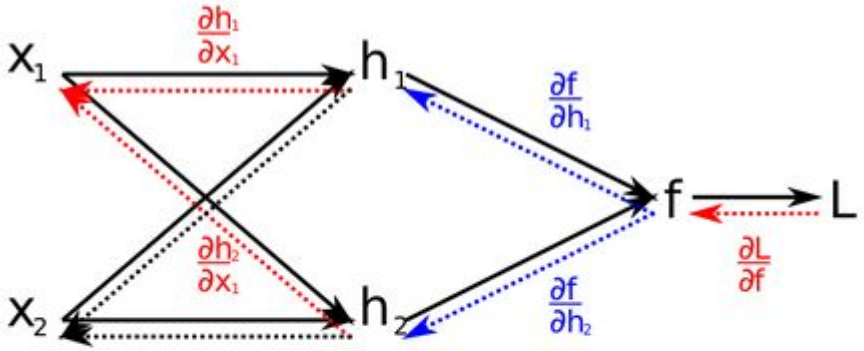
$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t)$$

donde $f$ es una función *diferenciable* que representa el objetivo de nuestra simulación (ej: reducir el error de predicción)

$\alpha$ es la tasa de aprendizaje

# Repaso redes neuronales

Propagación hacia atrás




Chain rule of derivatives:

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial x_1} = \frac{\partial L}{\partial f}\left(\frac{\partial f}{\partial h_1}\frac{\partial h_1}{\partial x_1} + \frac{\partial f}{\partial h_2}\frac{\partial h_2}{\partial x_1}\right)$$

# Repaso redes neuronales

Ventajas:

- Permiten resolver problemas de alta complejidad.
- En RL, son los únicos métodos que tienen un amplio funcionamiento en muchas tareas.

A tener en cuenta:

- Caja negra, poco interpretables.
- Suelen requerir mucha ingeniería y tiempo de ajuste.
- Pueden caer en máximos locales, o en inestabilidades.

# Algoritmos de deep RL

# Algoritmos de deep RL

Existen (a grandes rasgos) tres tipos de algoritmos model-free de deep RL usados en la actualidad...

1. Basados en valor (crítico): aproximan $\hat{Q}(s, a \mid \theta) \approx Q(s, a)$
2. Basados en política (actor): generan la política mediante $\pi(s, a \mid \theta)$
3. Actor-crítico: combinan ambos

# Algoritmos basados en valor

Los primeros algoritmos de deep RL construyen a partir de Q-Learning

# Algoritmos basados en valor

- Al igual que en Q-Learning, se basan en lograr la convergencia a partir de la predicción del valor V o acción-valor Q.
- El primer algoritmo, DQN, fue el que inició el aprendizaje por refuerzos profundo, combinando Q-Learning con redes neuronales profundas.

# Algoritmos basados en valor

Esta combinación tiene un problema! La **“Tríada Mortal”** 💀

Problema empírico causante de divergencias al combinar:

- Aproximación funcional.
- Aprendizaje off-policy.
- Bootstrapping.

# Algoritmo DQN

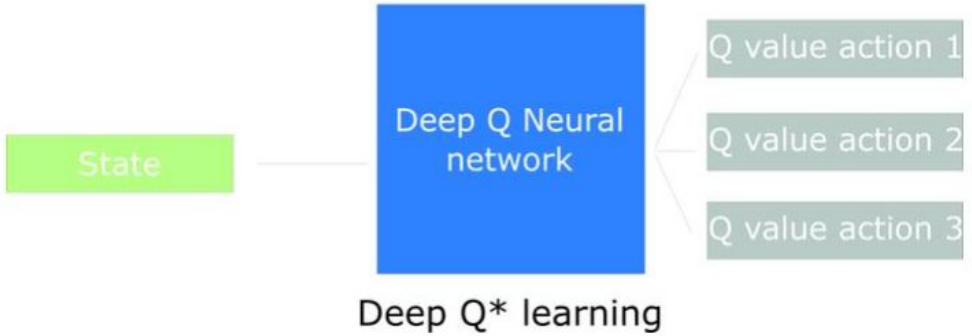
Primer algoritmo de deep RL: *Deep Q-Networks*. Se proponen varias mejoras respecto a Q-learning, manteniendo la base:

- Se selecciona la mejor acción con probabilidad $1 - \varepsilon$ , y una aleatoria con probabilidad $\varepsilon$
- Para determinar la mejor acción, evaluamos el valor $Q(s, a)$ (estimado) para cada acción, estando en el estado $s$

# Algoritmo DQN

Pasamos de ...


A ...



# Algoritmo DQN

El algoritmo incluye varias mejoras

Refrescamos antes la actualización de Q-Learning (de la clase pasada)

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \arg\max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

# Algoritmo DQN

1. Usar pesos $\phi$ para estimar $\hat{Q}$, actualizarlos mediante descenso de gradiente, usando la función de error

$$L(\phi) = (y - \hat{Q}(s, a \mid \phi))^2$$

en donde

$$y = \begin{cases} r & \text{si } s' \text{ es un estado final} \\ r + \gamma max_{a'} \hat{Q}(s', a' \mid \phi^-) & \text{en caso contrario} \end{cases}$$

¿Ahora bien, en base a qué experiencias actualizamos esos pesos?

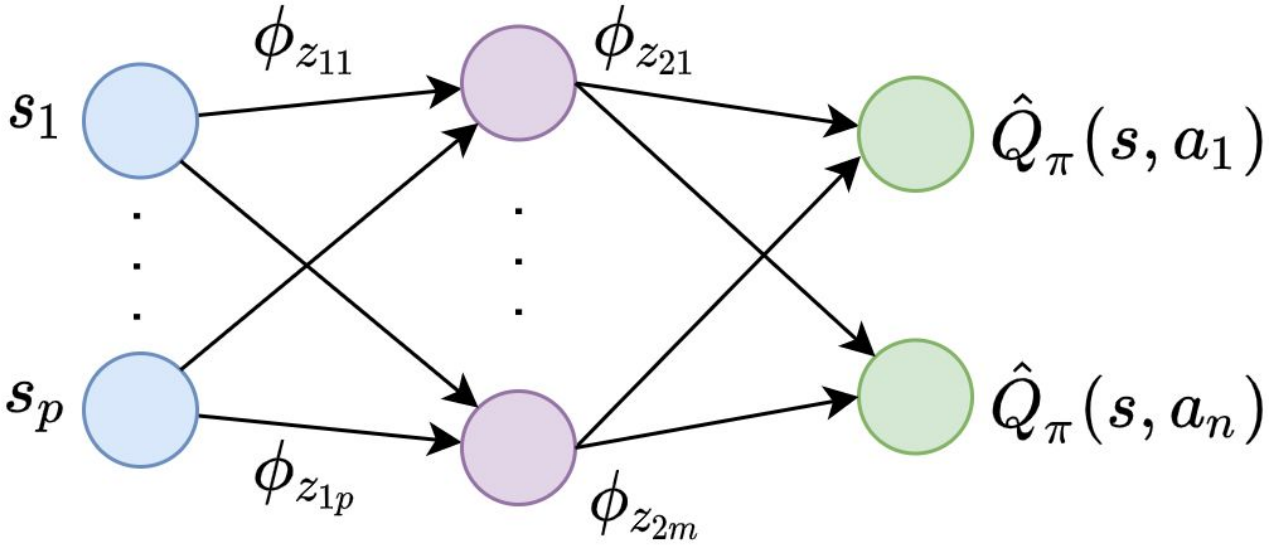
# Algoritmo DQN

2. Almacenamos cada transición $(s, a, r, s')$ en una memoria de experiencias.

3. Tras cada acción, muestreamos de esa memoria y usamos esas experiencias para actualizar los $\phi$.

# Algoritmo DQN: red neuronal

Red neuronal DQN de ejemplo (una capa oculta)...



# Algoritmo DQN

Problema: propenso a errores de sesgo: tienden a sobreestimar $\hat{Q}(s, a \mid \theta)$

Para evitar esto, las mejoras principales fueron en esa dirección.
Simplificadamente:

- Double DQN (se usa una segunda red Q para determinar la mejor acción en el siguiente estado de la actualización de Q)

# Algoritmo DQN

- Dueling DQN: se estima $A(s,a)$ y $V(s)$ , en lugar de $Q(s,a)$ , para poder aprender cuáles estados son “valiosos”, independientemente del efecto de cada acción. Esto es especialmente útil en aquellos estados donde la acción particular que se toma no es tan relevante.
- Prioritized experience replay: se muestrean acciones según la magnitud del error de actualización, pesada por su probabilidad de haber sido elegidas.

# Algoritmos de gradiente de política

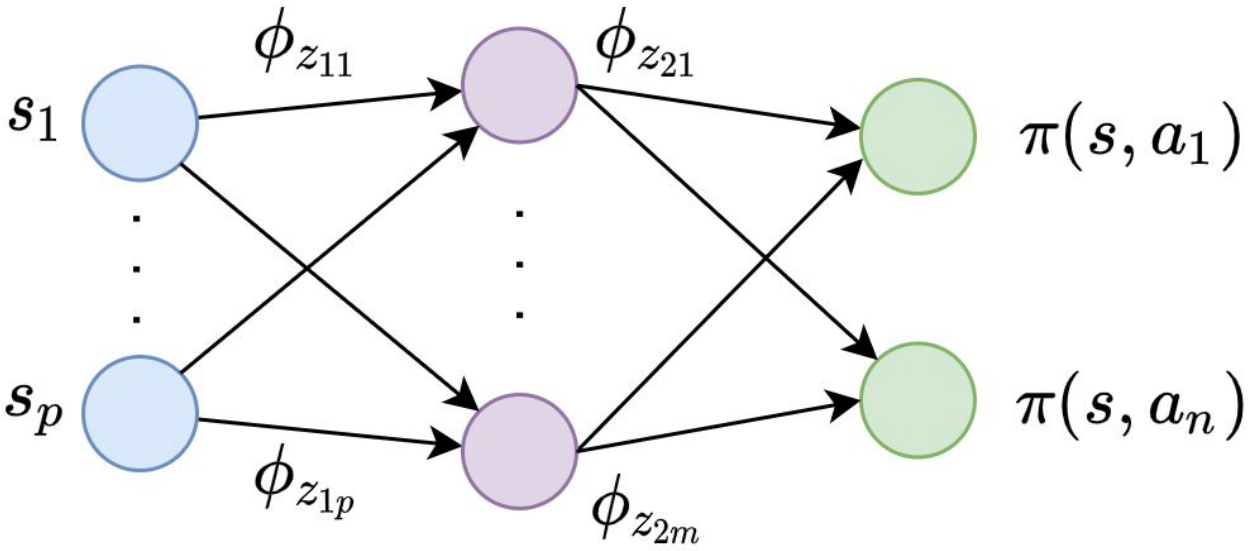
Otro enfoque de algoritmos de deep RL se basa en la aproximación directa de la política

# Algoritmos de gradiente de política

- En lugar de aprender $\hat{Q}(s, a \mid \theta)$ , aprender directamente la política $\pi(s, a \mid \theta)$ (expresada también como $\pi_\theta$ )
- Se recolectan experiencias con una misma $\pi(s, a \mid \theta)$ , se estiman los valores por promedio, y se actualiza $\theta$ en base a las experiencias.

# Algoritmos de gradiente de política

Ejemplo de red básica que implementa esto. ¿Cómo actualizamos los pesos?



# Gradiente de política

El objetivo al aprender la política $\pi_\theta$ es, dada una trayectoria $\tau = (s_0, a_0, \ldots, s_{t+1})$, maximizar el retorno esperado

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$$

para ello, podemos optar por actualizar $\theta$ en base al valor anterior, de acuerdo a

$$\theta_{t+1} = \theta_t + \alpha J(\pi_\theta)\Big|_{\theta_t}$$

# Gradiente de política

Esta actualización de pesos se puede hacer mediante **ascenso de gradiente** por medio del teorema de gradiente de política, que establece que:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_{t=1}^{T}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)]R(\tau)]$$

# Gradiente de política

El valor esperado puede ser aproximado con $D$ trayectorias, quedando

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} [\nabla_\theta \log \pi_\theta(a_t \mid s_t)] R(\tau)$$

# Algoritmos de gradiente de política

- Los algoritmos de gradiente de política originaron en los 90’ con *REINFORCE*, y se incorporaron recientemente al deep learning.
- Ventaja: la función $\pi$ se actualiza de acuerdo a la experiencia muestreada; no es tan indirecta como lo era con $\varepsilon$-greedy

(la probabilidad según $\varepsilon$-greedy era:

$$\pi(s,a) = \begin{cases} 1-\varepsilon & \text{si es la mejor accion segun su } Q(s,a) \\ \varepsilon & \text{caso contrario} \end{cases}$$

# Algoritmos de gradiente de política

Problema: actualizar de esta forma es propenso a errores de varianza, pudiendo “romper” políticas si $\theta_{t+1}$ cambia demasiado de $\theta_t$

- Solución 1: limitar cuánto puede variar la nueva política. Implementado por algoritmo *Trust Region Policy Optimization*.
- Solución 2: usar una métrica más estable que el retorno de las trayectorias. Implementado por algoritmos actor crítico...

# Algoritmos actor-crítico

Los algoritmos AC combinan ventajas tanto de la generación de políticas (actor) como de la estimación del valor (crítico)

# Algoritmos actor-crítico

Los algoritmos AC entrenan 1) función $\pi(s, a)$ para seleccionar acciones, 2) un crítico $\hat{V}(s)$, que estima el valor de los estados.

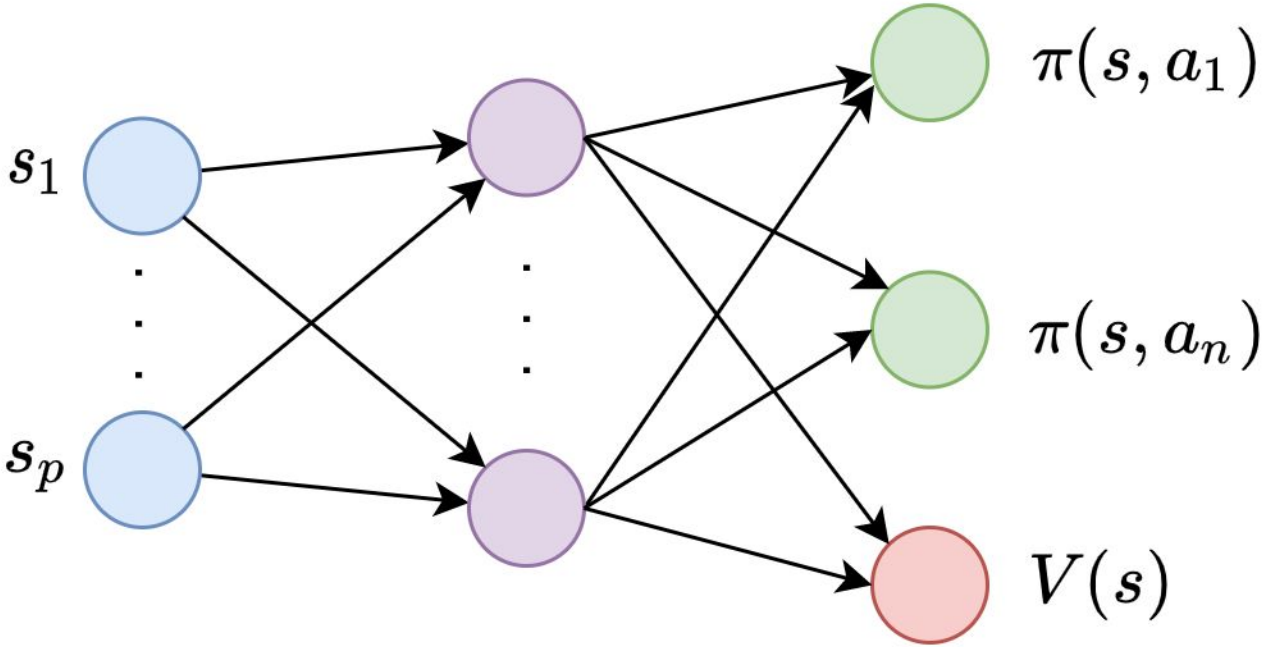
Ventajas:

- El actor selecciona acciones.
- El crítico confiere estabilidad (reemplaza el promedio de los retornos).

Vemos algunos de los más relevantes actualmente.

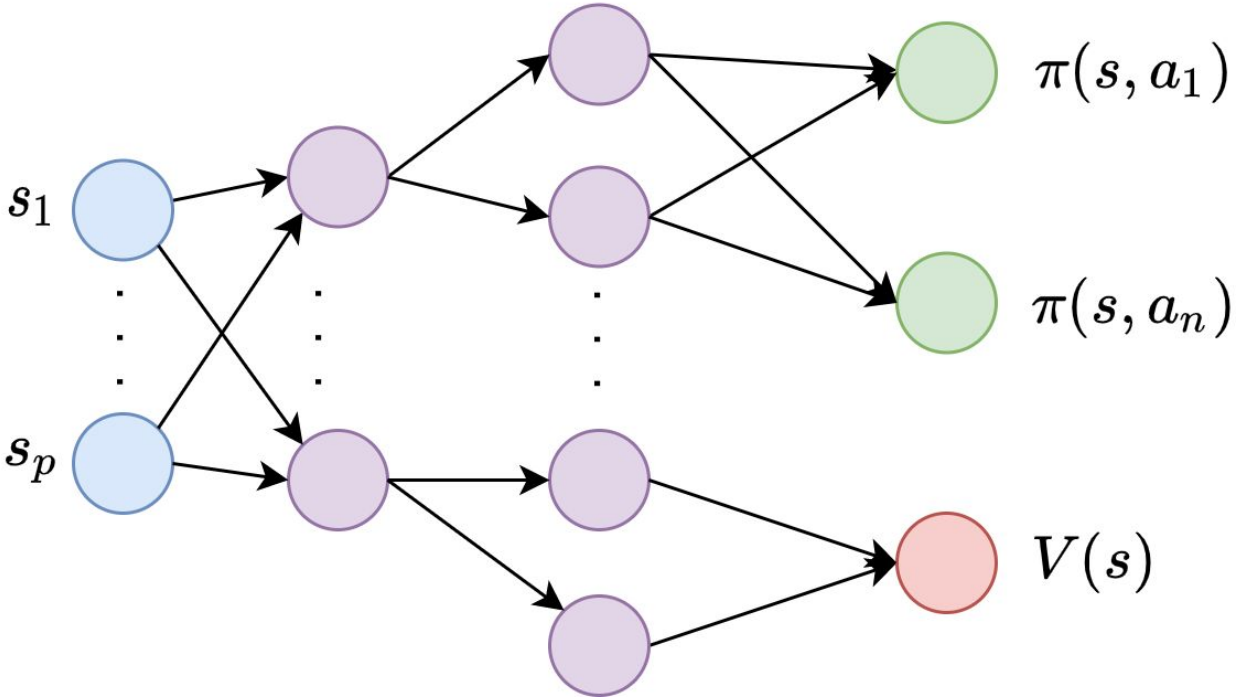
# Algoritmos actor-crítico

La red ahora pasa a tener una estructura así...



# Algoritmos actor-crítico

También puede dividirse en subred de actor y de crítico...



# Algoritmos actor-crítico

- Los pesos del actor se actualizan mediante gradiente de política.
- Los pesos del crítico se actualizan según error de diferencia temporal.
- Vemos dos algoritmos de ejemplo...

# Algoritmo DDPG

- DQN adaptado como actor-crítico para manejar acciones continuas.
- Selecciona las acciones al muestrear el actor determinístico $a(s \mid \theta) + \varepsilon$, siendo $\varepsilon$ un ruido de exploración
- En espacios de acciones continuas, es muy difícil encontrar $\max_a Q(s, a)$
- En cambio, se buscar aproximar mediante $\max_a \approx Q(s, a(s, \theta))$ (con ello actualiza el crítico; el actor se actualiza con gradiente de política).
- La actualización de los pesos se realiza como en DQN, a través de experiencias pasadas.

# Algoritmo PPO

Algoritmo estrella de gradiente de política

- Combina AC con un mecanismo para limitar cuánto pueden variar las nuevas políticas, mejorando el de TRPO.
- Permite trabajar con acciones continuas y discretas.
- Desventaja: es on-policy, con lo cual no puede usar buffer de experiencias

# Ejemplos de aplicaciones de deepRL

# Deep RL en Juegos

La pasada clase vimos que RL explotó en los juegos

Vemos ahora algunas de estas arquitecturas a mayor detalle...

# Juegos

Atari

Deepmind

Paper

Juegos/links al código

# Juegos

Atari

- Extracción de características con redes convolucionales.
- Primer algoritmo que combinó RL con *deep learning*: *DQN.*
- Muy buen desempeño en la mayoría de los juegos.
- La suite de Atari se vuelve el primer *benchmark* de prueba de deep RL.
- Introduce varias asunciones: recortes en recompensas, apilamiento de imágenes de entrada.

# Juegos

Go

Deepmind

Paper

Link a LeelaZero

(replicación

open source)

# Juegos

AlphaGo

- Integra monte carlo tree search (Chaslot, 2008) con redes neuronales profundas y self-play.
- Profundiza los avances de la década pasada en Go por computadora y logra vencer al gran maestro de Go, Lee Sedol en una exhibición.
- Utiliza partidas pasadas como forma de pre-entrenamiento.
- Profundiza los avances de la década pasada en Go por computadora y logra vencer al gran maestro de Go, Lee Sedol en una exhibición.

# Juegos

- AlphaGo fue sucesivamente refinado en AlphaGoZero, AlphaZero y MuZero para quitar progresivamente ciertas asunciones.
- Para ver en más detalle los mismos pueden ver este post, o este video de Henry Labs (la imagen de la slide siguiente es del mismo).

# Juegos

Timeline Overview

1. **AlphaGo** → Monte Carlo Tree Search using 3 Convolutional Policy Networks, 2 of which are trained to copy expert moves, another with policy gradients, and a separate Value Network.
2. **AlphaGo Zero** → No Supervised Learning of expert moves, Policy and Value networks combined into a single Residual Neural Network, Policy Net updated to match MCTS actions
3. **AlphaZero** → Restructure input and output representations to play Chess and Shogi as well, restructure self-play algorithm
4. **MuZero** → Remove assumption of a given dynamics model, introduce hidden state in order to do MCTS with a learned dynamics model, starting from a learned root state initialization

# Juegos

- Como varios de los hitos de esta magnitud, insumió un muy alto costo energético para el entrenamiento! (estimación)
- De ahí empezaron a surgir muchos debates sobre la comunidad sobre esto...

# Juegos

OpenAI Five (Dota 2)

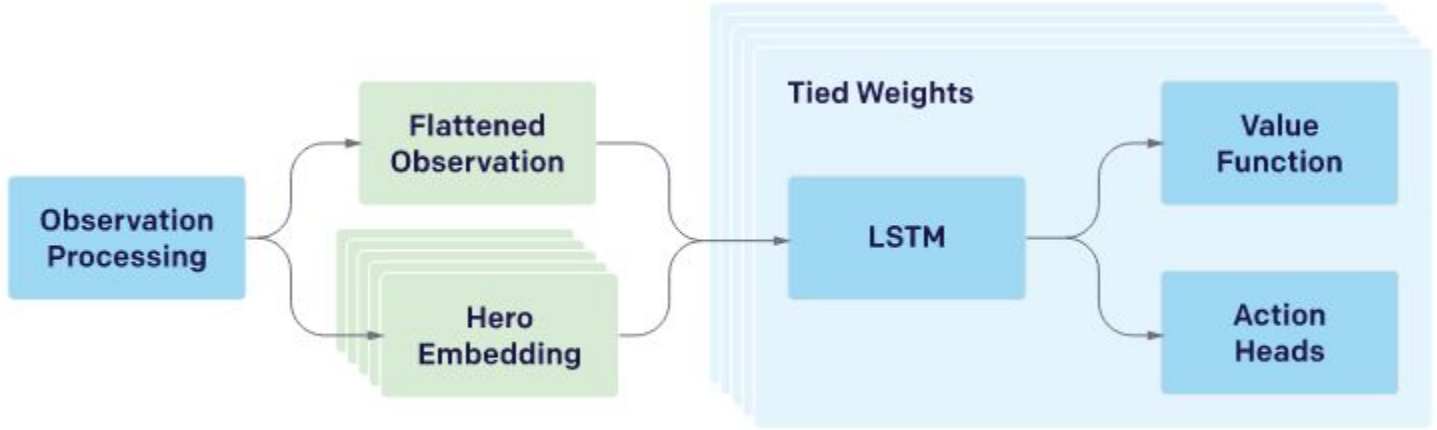
OpenAI

Paper

# Juegos

OpenAI Five

- Combina algoritmo PPO con cinco redes neuronales masivas que deciden las acciones de cada jugador. Simplificadamente (fuente: paper de Dota 2)



# Juegos

OpenAI Five

- Se entrena a partir de self-play
- Obtiene el estado a partir de una API.
- Usa una versión del juego reducida en complejidad.
- Importante: recompensas diseñadas a mano para guiar los agentes a los objetivos.

# Juegos

Dota 2

Función de recompensa ***MUY*** artesanal

| Name | Reward | Heroes | Description |
|---|---|---|---|
| Win | 5 | Team | |
| Hero Death | -1 | Solo | |
| Courier Death | -2 | Team | |
| XP Gained | 0.002 | Solo | |
| Gold Gained | 0.006 | Solo | For each unit of gold gained. Reward is not lost when the gold is spent or lost. |
| Gold Spent | 0.0006 | Solo | Per unit of gold spent on items without using courier. |
| Health Changed | 2 | Solo | Measured as a fraction of hero's max health.‡ |
| Mana Changed | 0.75 | Solo | Measured as a fraction of hero's max mana. |
| Killed Hero | -0.6 | Solo | For killing an enemy hero. The gold and experience reward is very high, so this reduces the total reward for killing enemies. |
| Last Hit | -0.16 | Solo | The gold and experience reward is very high, so this reduces the total reward for last hit to $\sim 0.4$. |
| Deny | 0.15 | Solo | |
| Gained Aegis | 5 | Team | |
| Ancient HP Change | 5 | Team | Measured as a fraction of ancient's max health. |
| Megas Unlocked | 4 | Team | |
| T1 Tower* | 2.25 | Team | |
| T2 Tower* | 3 | Team | |
| T3 Tower* | 4.5 | Team | |
| T4 Tower* | 2.25 | Team | |
| Shrine* | 2.25 | Team | |
| Barracks* | 6 | Team | |
| Lane Assign† | -0.15 | Solo | Per second in wrong lane. |

# Juegos

También es artesanal el ajuste de hiper-parámetros (img. del paper de Dota 2)


Figure 32: **Learning Rate during The International:** This is what happens when humans under time pressure choose hyperparameters. We believe that in the future automated systems should optimize these hyperparameters instead. After this event, our team began to internally refer to the act of frantically searching over hyperparameters as "designing skyscrapers."

# Juegos

Muy alto costo energético:

- Para entrenar el agente de OpenAI que jugó al Dota 2 se necesitaron correr 256 GPUs y 128.000 núcleos de CPU durante varias semanas. entrenando por día el equivalente a 180 años.

# Juegos

AlphaStar (Starcraft 2)

Deepmind

Paper

Github con experimentos pequeños

# Juegos

AlphaStar

- Adapta self-play de forma compatible con la estrategia de Starcraft 2.
- Involucra mecanismos de *imitation learning* para iniciar el entrenamiento a partir de partidas de grandes maestros (ej: aprender aperturas).

# Juegos

AlphaStar

Tras inicializar, se entrenan los agentes con un algoritmo que maximiza su % de victorias frente a una liga de agentes elegido mediante algoritmos multi-agente, para converger a un equilibrio de Nash. Oponentes de la liga:

- Agentes “meta” que juegan de la forma que más prob. de victoria tienen.
- Agentes “explotadores” que apuntan a las estrategias “anti-meta”.
- “Explotadores de liga”, que buscan explotar las debilidades de toda la liga.

# Juegos

AlphaStar

- Al igual que para Dota 2, se usan redes recurrentes para manejar la observabilidad parcial.
- Usa una versión sin restricciones del juego.
- Presenta varios problemas, que muestran en cierto punto las limitaciones de este enfoque.

# Juegos

Costo energético (muy alto, al igual que Dota 2 y AlphaGo)

- Según el paper de AlphaStar, se usaron: 32 TPUs (3era gen) en 44 días

# Más allá de los juegos

Los juegos son buenos para probar nuevos enfoques y ver el funcionamiento y limitaciones con entornos acotados.

No obstante, lo ideal es usar esos enfoques en aplicaciones más abarcativas.

Vemos algunas aplicaciones variadas.

# Recomendadores

Spotify usa aprendizaje por refuerzos para diversificar recomendaciones de temas musicales, dada una representación de la sesión del usuario

$$R(t, s) = r(t, u) - c + \alpha d\,(t, u) r(t, u)$$

Algoritmo: REINFORCE

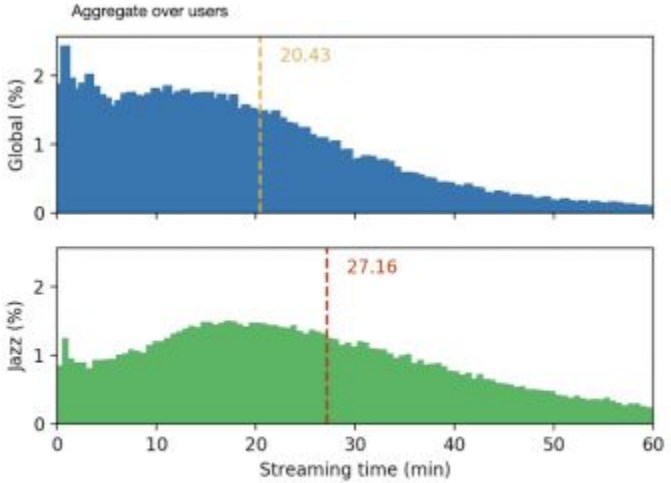
$t \sim \pi(\cdot|s)$ (t es el tema musical, s es la sesión del usuario u)

$r \in \{0,1\}$ (0 si el usuario saltó el tema, 1 en caso contrario)

$d(t,u) \in [0,1]$ (cuantifica la diversidad del tema)

# Recomendadores

- También usa algoritmos de bandidos para adaptar esa función $r \in \{0,1\}$ en ciertos contextos, según el tipo de usuario y la lista de reproducción.


Jazz listeners consume Jazz and other playlists for longer period than average.

# Control

Three Springs Tech
(control de rueda recogedora de materiales)

# Control

Three Springs Tech

- Utilizan RL para controlar una rueda recogedora.
- Usan algoritmo DDPG debido a las acciones continuas en el control de la rueda.
- Función de recompensa cuyo máximo es la carga del 100%, pero castiga la sobrecarga mucho más que la sub-carga.

# Control

Algoritmo: DDPG

Estado

- Perfil que representa lo que hay delante de la rueda.
- Presión hidráulica de la rueda.
- Indicador de la carga que está recogiendo.

Acciones

- Velocidad de giro.
- Velocidad de movimiento de la rueda.

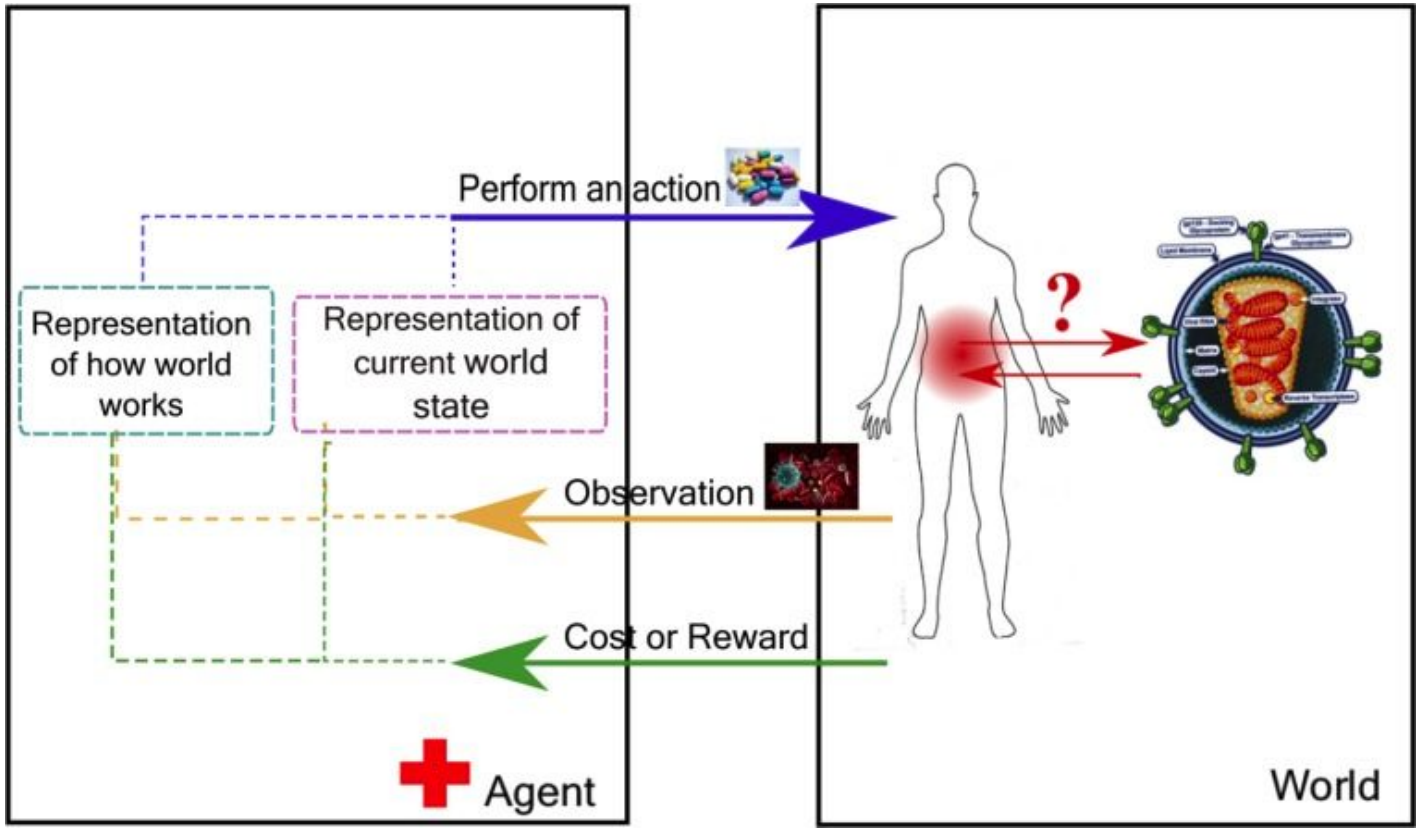
# Salud

Tratamiento de pacientes con HIV (fuente)

- Objetivo: incrementar la eficacia en las terapias de tratamientos con drogas antirretrovirales en pacientes con HIV.
- Terapia: combinación de drogas antirretrovirales suministradas al paciente. Estos tratamientos son a largo plazo, y dependen de variables como la carga viral del paciente, y su sistema inmune.

# Salud

Enfoque de RL propuesto



# Salud

Estado

- Estado fisiológico real del paciente es desconocido. Se estima a partir de una observación de sus defensas, carga viral y mutaciones del virus.

Acciones

- Aplicar una terapia antirretroviral en tiempo *t,* a partir de combinaciones de drogas.

Recompensa

- En función de si la terapia tiene efecto, ej.: si bajó la carga viral.

# Salud

- El agente se entrena a partir de una base de datos pública, anonimizada y aportada voluntariamente por >100.000 pacientes y sus tratamientos del HIV.
- Se toma a cada paso de tiempo como una instancia de aplicación de terapia y sus correspondientes mediciones posteriores.

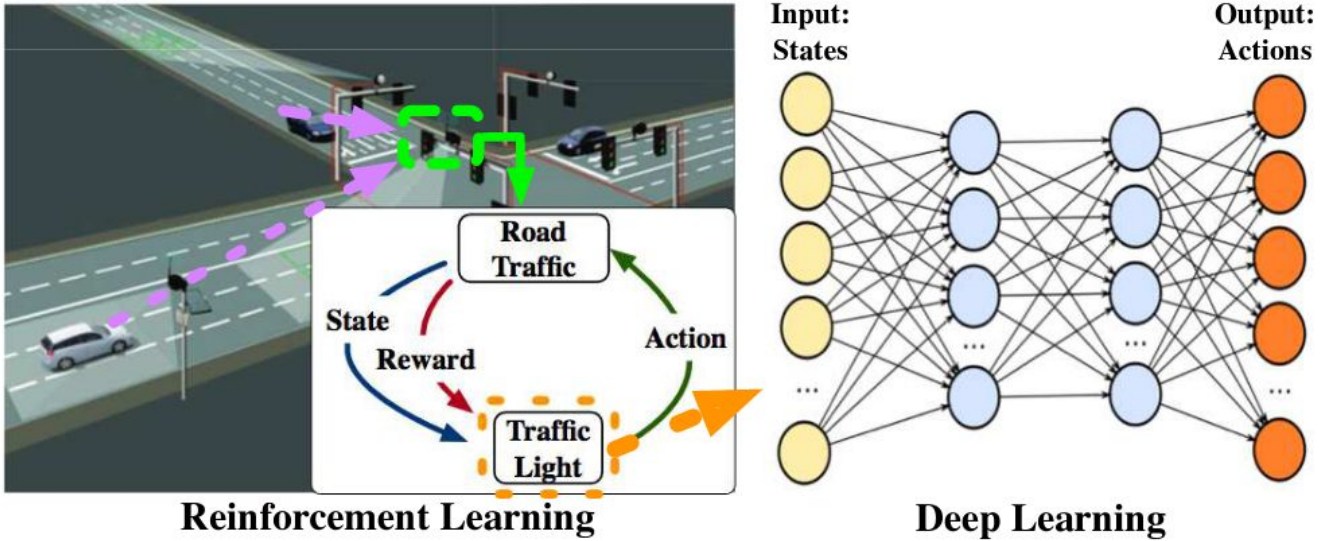
# Salud

- Los autores reportan resultados promisorios de combinar este enfoque con agrupamiento de pacientes similares, permitiendo obtener predicciones sobre cómo funcionará un determinado tratamiento.
- Dado que involucramos pacientes, el uso de este tipo de aplicaciones debe supervisarse por expertos y considerarse como una forma de refinar tratamientos existentes (fuente).

# Control de tráfico

Control de semáforos (fuente 1, 2)

- Puede modelarse en única intersección o en una red de semáforos.



# Control de tráfico

Estado

- Involucra variables como la posición en la calle de cada auto o la cantidad de autos en cola. Se procesan con redes neuronales y a partir de imágenes u otras características.

Acciones

- Involucran cambiar (o mantener) las fases en la dirección de tráfico.

Recompensa

- Relacionada a la demora total, frenadas, flujo de tráfico.

Pasos de tiempo suelen llevar “fracciones” como 0.5, 1 o 5 segundos.

# Calendarización y simulaciones de escenarios

# Transferencia a entornos reales (sim2real)

Entrenar modelos en simulación es barato

Obtener experiencia de entornos reales es muy costoso

¿Cómo transferimos lo entrenado para que funcione en un entorno real?

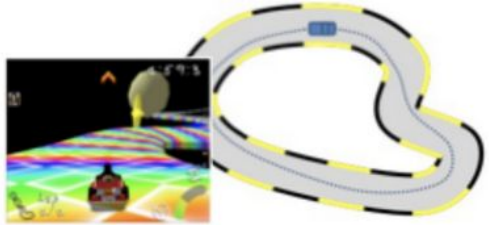
# Transferencia a entornos reales

Algunas técnicas:

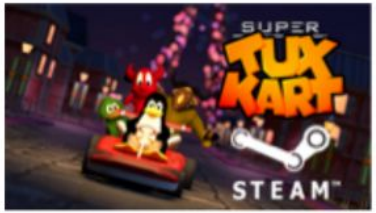
- Imitation learning
- Domain adaptation
- Domain randomization

# Transferencia a entornos reales

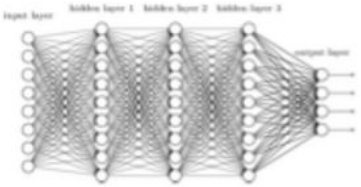
Imitation learning (fuente)



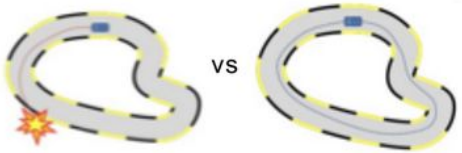
Demonstrations or Demonstrator



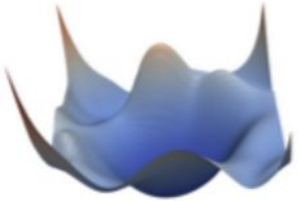

Environment / Simulator



Policy Class




Loss Function



Learning Algorithm

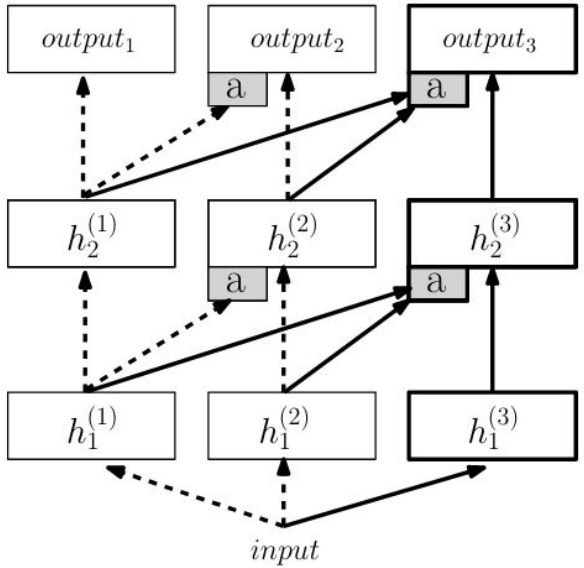
# Transferencia a entornos reales

- Problema con imitation learning: “distributional shift”: se asume que los datos son IID cuando en realidad dependen de muchas interacciones anteriores.
- No obstante, usarlas para pre-entrenar y luego usar un algoritmo de RL tradicional puede resultar de mucha utilidad.

# Transferencia a entornos reales

Domain adaptation

- Se busca adaptar un entrenamiento realizado en un contexto (ej.: simulación) a otro (ej.: entorno real).
- Ejemplo: *progressive neural networks*:

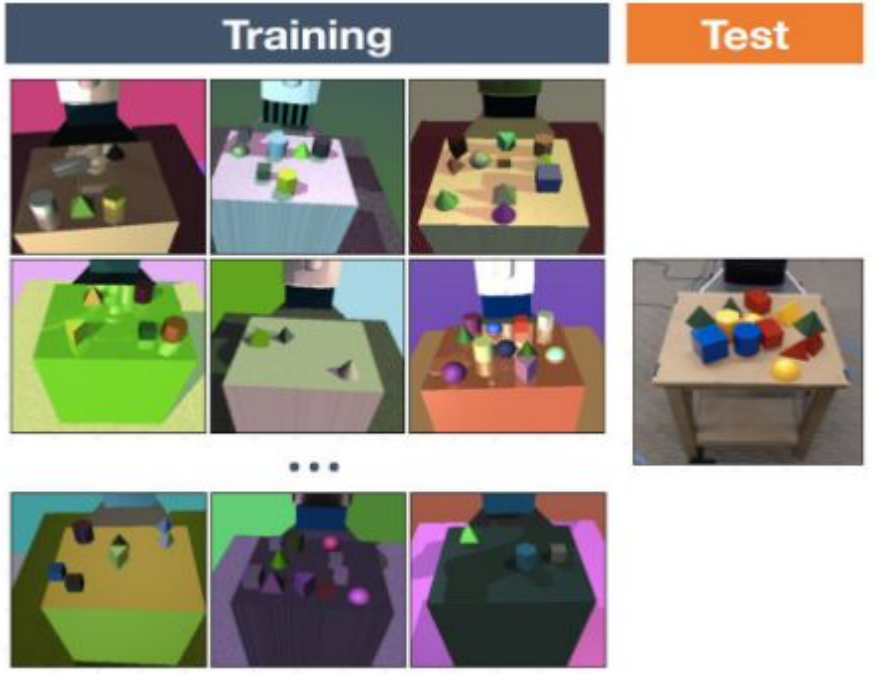


# Transferencia a entornos reales

Domain randomization

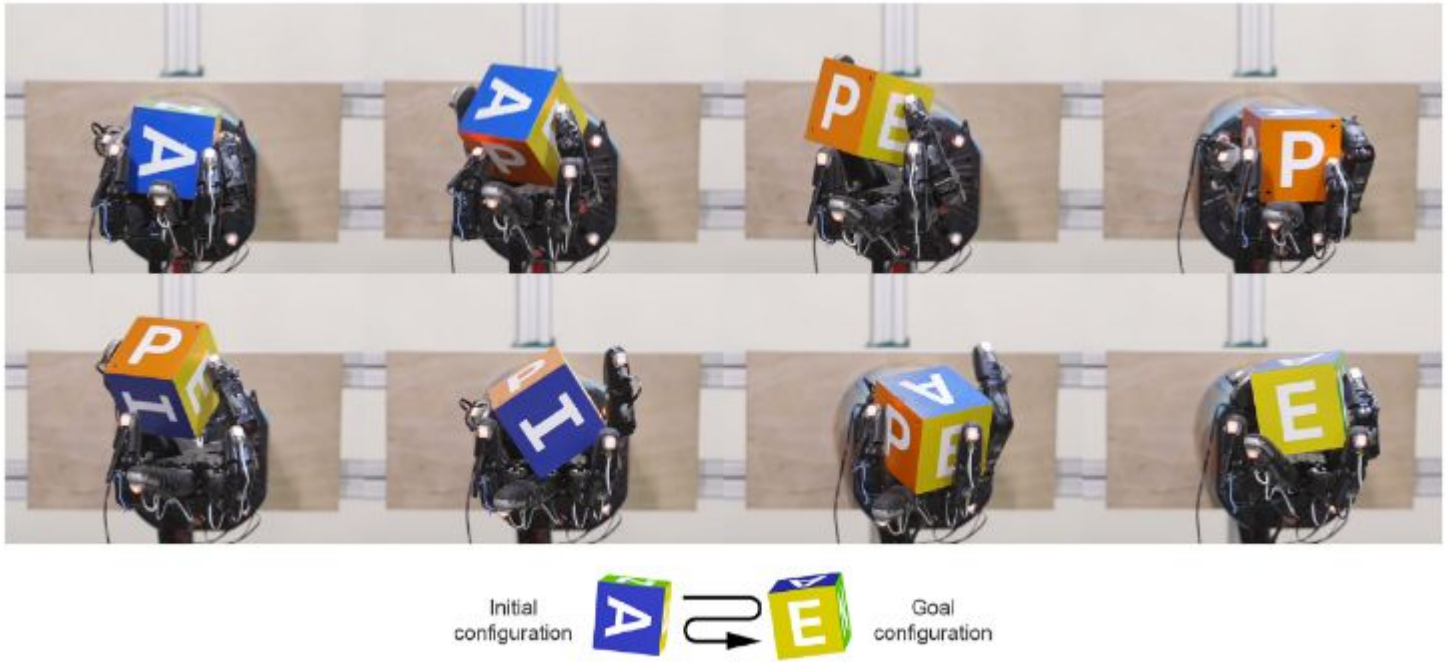
- El entorno real es considerado como una variante más

Fuente



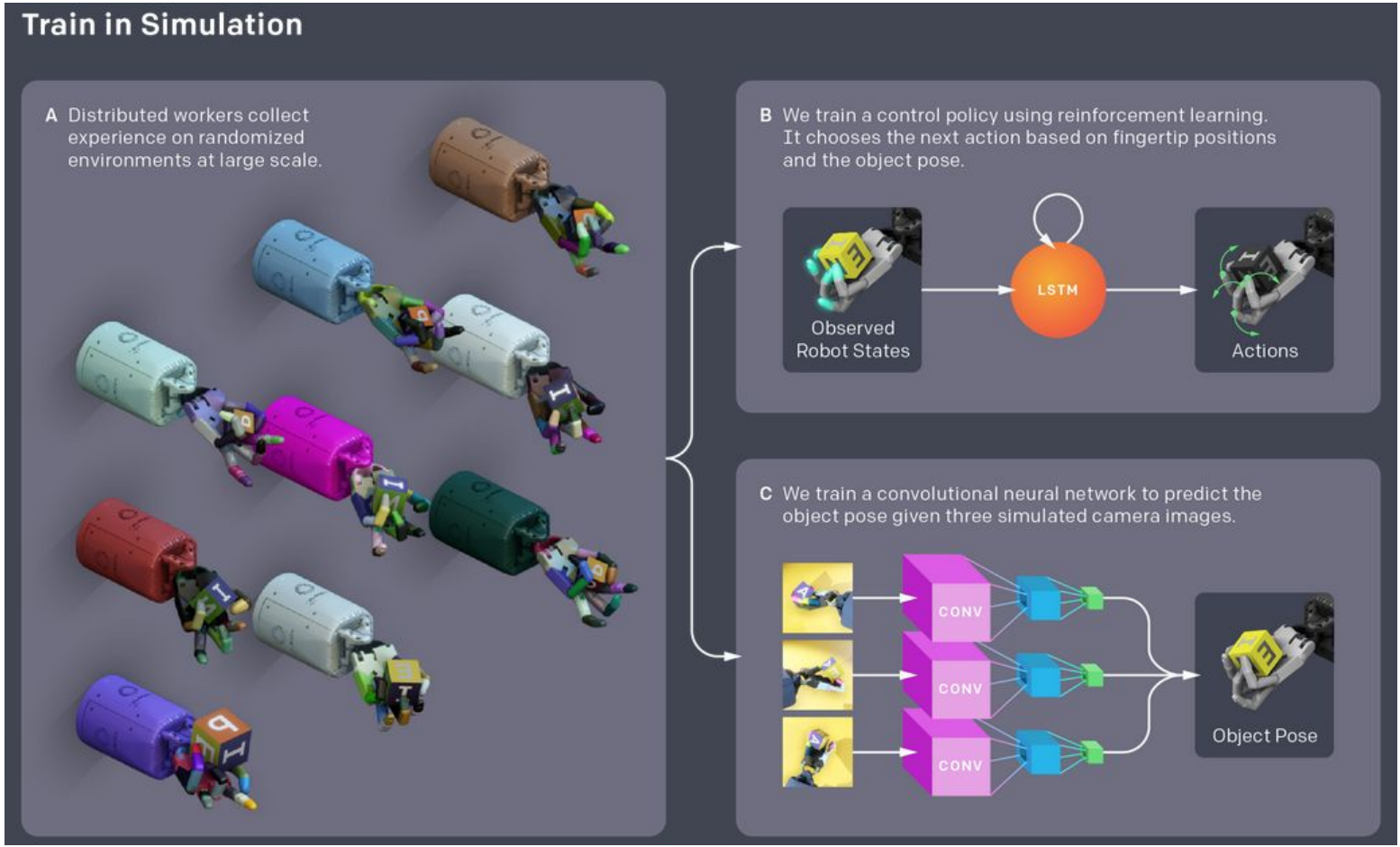
# Robótica

- Manipulación de manos (paper, blog)



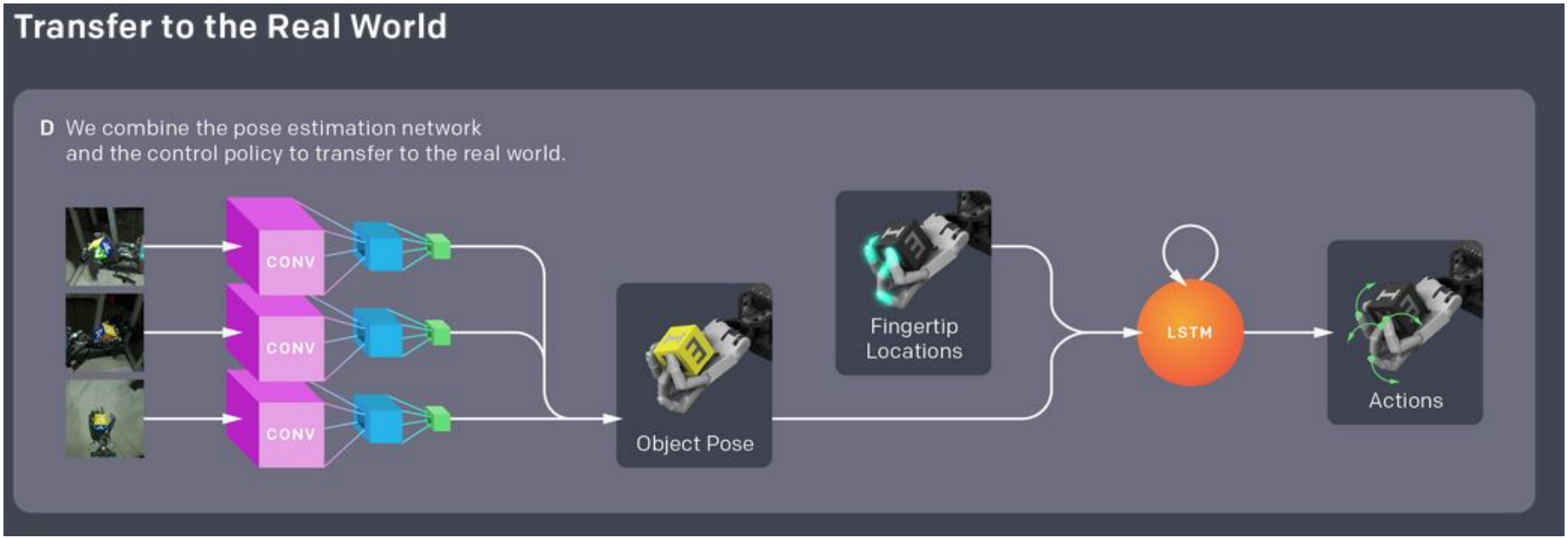
# Robótica

Fuente



# Robótica

Fuente



# Robótica

Aplicación en industria:

Covariant.ai

# Robótica

Covariant

- Apunta a automatizar tareas industriales que requieren manos (ej.: recoger y soltar productos).
- Estos robots están actualmente funcionando en una planta en Alemania.
- Para ello entrenan una “red neuronal gigante” *(sic)* en distintos productos con distintas unidades de recogida.

# Robótica

- Mezclan aprendizaje supervisado, imitation learning, few-shot learning y RL.
- Usan entrenamiento entre simulaciones y evaluación real (en robots y luego en el piso de planta).

# Conducción autónoma

Paper: Simulation-Based RL for  Real-World Autonomous  Driving

- Los autores entrenaron un agente RL en simulación usando el simulador open source CARLA (imagen).
- Luego, ponen a correr el agente en una calle real, con el mismo recorrido.
- Estado: imágenes e info adicional (ej: dirección de calle).
- Acciones: controlan dirección del volante (la velocidad se usa constante).

# Conducción autónoma

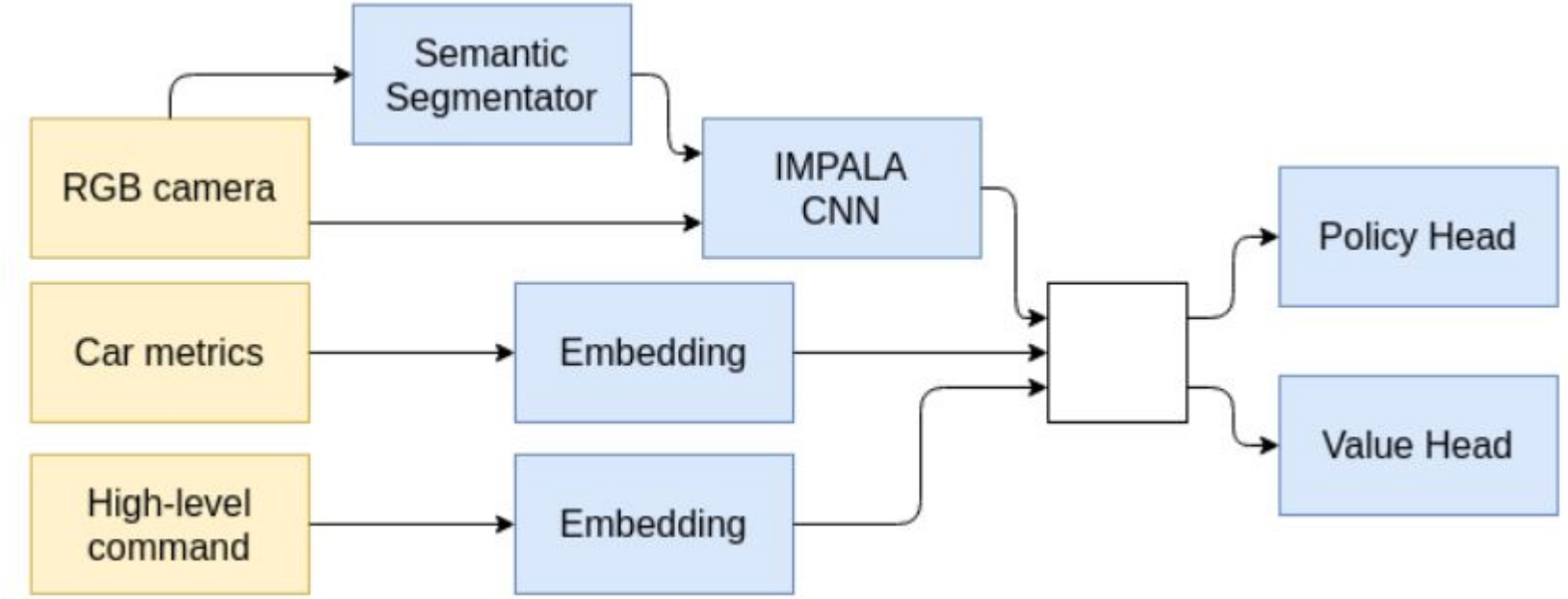
- Recompensa definida por seguir la trayectoria especificada.
- Cada episodio finaliza si el agente diverge 5 metros de la ruta, o choca contra un obstáculo (en la simulación). En la calle, finaliza cuando el conductor humano de seguridad debe intervenir.

# Conducción autónoma

- Se usa *domain randomization* que incluye variaciones de clima, distinta calidad de imágenes del simulador y ruido aleatorio introducido en ellas.
- Se entrena además un modelo de segmentación semántica (mediante aprendizaje supervisado) para detectar los objetos de las imágenes (CARLA provee las etiquetas de los mismos).

# Conducción autónoma

Arquitectura de NN usada



# Conclusiones

# Conclusiones

- El uso de técnicas de RL se incrementó significativamente en los últimos años.
- Hasta aquí abordamos un paneo general de algunas de ellas.
- RL también es usado en otros dominios que no llegamos a cubrir, como gestión de energía eléctrica y finanzas.
- En finanzas en particular, hay mucho debate sobre su utilidad - o no - en RL.

# Conclusiones

Algunos puntos a mejorar en RL (focos actuales de investigación en la comunidad)

- Eficiencia en el aprendizaje por cada muestra recolectada, aquí RL tiene todavía camino por mejorar.
- Técnicas de transferencia de aprendizaje y meta-aprendizaje.
- Ajuste automático de hiper-parámetros considerando las particularidades de RL.

# Conclusiones

- El objetivo de visualizar varias aplicaciones era el mostrar distintas ideas y enfoques, en una variedad de aplicaciones.
- En nuestros problemas se recomienda partir desde enfoques simples, e ir aumentando la complejidad a medida que vamos logrando una base de convergencia.

# Conclusiones

Algunas áreas promisorias a observar...

- Offline RL
- Meta-learning
- Curriculum learning
- Model-based RL

# Actividad 2: Programar con stable-baselines!

Link al notebook del Lab 2