Transferencia de Conocimiento Utilizando Múltiples Tareas en Algoritmos de Aprendizaje por Refuerzo Profundo

Jesús García-Ramírez, Eduardo Morales, Hugo Jair Escalante

Reporte Técnico No. CCC-19-003
Julio 2019

Coordinación de Ciencias Computacionales
Instituto Nacional de Astrofísica Óptica y Electrónica

Luis Enrique Erro 1
Sta. Ma. Tonantzintla,
72840, Puebla, México

Resumen

Las principales limitaciones de los métodos basados en aprendizaje profundo son que se necesita un conjunto de datos grande para que se obtenga un desempeño aceptable, además de que el tiempo de entrenamiento es demasiado si no se cuenta con equipo especializado. La transferencia de aprendizaje puede ser una solución para reducir el tiempo de entrenamiento en los enfoques basados en aprendizaje profundo. Los trabajos sobre este tema hacen énfasis en entrenar un modelo más pequeño a partir de uno pre-entrenado que tiene un buen desempeño en dicha tarea. Sin embargo, no se centran en realizar transferencia a una nueva tarea. En este reporte técnico se presenta una propuesta de investigación para realizar transferencia de conocimiento en el dominio de aprendizaje por refuerzo profundo utilizando múltiples tareas base. Elegir entre todas las un conjunto de tareas, cuál servirá para realizar la transferencia es un problema que se aborda en este reporte. Para solventar este problema se propone una medida basada en similitudes visuales y una comparación de espacios de acciones entre tareas. Para validar la medida propuesta se realizaron experimentos utilizando *fine-tuning*, donde se reutilizan los pesos de modelos pre-entrenados en las tareas base, haciendo transferencia entre una tarea base y una tarea objetivo.

Palabras Clave: Aprendizaje profundo, Aprendizaje por refuerzo, Aprendizaje por Refuerzo Profundo, Transferencia de Conocimiento

Abstract

One of the main limitations of deep learning approaches is the large number of examples in order to obtain an acceptable performance, and that the training time is computational expensive even with specialized hardware. Transfer learning approaches offer an alternative to reduce the training time in deep learning approaches. Most of the research in transfer learning for deep reinforcement learning are based on training a smaller model (that means lower parameters in the model) using an expert model to obtain acceptable performance in a certain task. Few works are based on transfer knowledge to a new task. In this work a method to transfer knowledge to a new task using multiples source tasks is presented. Selecting the best source task for transfering knowledge is an important problem to solve in this work. We propose a measure based on visual similarities and the differences in the action spaces of the source and target tasks. To validate the performance of the metric we perform tests with a fine-tuning approach in convolutional neural networks, reusing a pre-trained model using one source task and one target task.

Keywords: Deep Learning, Reinforcement Learning, Deep Reinforcement Learning, Transfer Learning

Contenido

1 Introducción

En años recientes la inteligencia artificial ha tenido un gran auge dentro de la comunidad científica. En particular los algoritmos de clasificación (regresión) han sido muy utilizados debido a su desempeño en tareas difíciles hasta hace algunos años.

Los clasificadores basados en aprendizaje profundo [LeCun et al., 2015] (*Deep Learning, DL*) han mostrado tener buen desempeño en tareas complejas como: estimación de pose [Güler et al., 2018], clasificación de imágenes [Krizhevsky et al., 2012], aprender a jugar vídeo juegos con un espacio de estados muy grande de instancias [Silver et al., 2017, Silver et al., 2016], entre otros. Las principales ventajas del DL son que se extraen características de manera automática, evitando en muchas ocasiones un paso previo de pre-procesamiento en los datos y que utilizan de entrada datos en bruto. Existen dos grandes limitaciones de estos enfoques, que son las siguientes: se necesita una gran cantidad de datos para tener un buen desempeño, esto debido al gran número de parámetros que se manejan; además de que se necesita equipo especializado para reducir el tiempo de entrenamiento.

En el aprendizaje por refuerzo (*Reinforcement Learning, RL*), se busca que un agente aprenda de manera autónoma mediante la experiencia, donde se interactúa con una representación del ambiente, utilizando una función de recompensa que puede ser positiva o negativa dependiendo de la decisión que tome el agente [Sutton, 2018]. La combinación entre DL y RL se ha nombrado aprendizaje por refuerzo profundo (*Deep Reinforcement Learning*, DRL). Estos algoritmos se utilizan cuando el espacio de estados y acciones es muy grande (por ejemplo, juegos de Atari) y se basan en actualizar los pesos de una red neuronal profunda para maximizar la recompensa obtenida. Los algoritmos de DRL pueden predecir la política (π, como se va a comportar un agente), el valor Q (recompensa a largo plazo) o ambos (métodos tipo *actor-critic*), además de aprovechar las ventajas de DL.

Los algoritmos basados en DRL aprovechan las ventajas de ambos enfoques. Sin embargo, también tienen sus limitaciones como la gran cantidad de datos para aprender alguna tarea con un desempeño aceptable. La gran ventaja es que estos algoritmos han obtenido mejor desempeño en tareas complejas, en ocasiones obteniendo mejor que los seres humanos. Como se muestra en [Mnih et al., 2015] donde se aprende una estimación del valor Q de las acciones disponibles en el dominio de juegos de Atari.

Una forma de reducir el tiempo de entrenamiento es la transferencia de conocimiento. Donde a partir de un conjunto de tareas base ($\mathcal{T} = t_1, t_2, t_3, \ldots, t_n$), se transfieren elementos relevantes a una tarea objetivo t_o (instancias, pesos, arquitectura, etc.). Esta técnica es utilizada cuando en la tarea objetivo no se cuentan con ejemplos suficientes obtener un desempeño aceptable. La principal desventaja es la transferencia negativa, este es el caso donde se obtiene un desempeño peor al que si se entrenará el modelo desde cero. En nuestro caso, el objetivo es que un agente DRL aprenda en menos tiempo.

Un problema que aún sigue abierto es definir cuál de las tareas del conjunto $\mathcal{T}$ tendrá un mejor desempeño cuando se transfiera para el entrenamiento de t_o. La adecuada selección de éstas tareas puede reducir el riesgo de transferencia negativa, así como el tiempo de entrenamiento.

Los métodos actuales de transferencia de aprendizaje en DRL se basan en utilizar un modelo previamente entrenado en una tarea y aprender ésta con un modelo más pequeño (modelo con menos parámetros por ajustar), este enfoque es llamado *policy distillation* [Rusu et al., 2015, Parisotto et al., 2016, Yin and Pan, 2017]. La evaluación de los modelos entrenados con redes neuronales no es computacionalmente costosa, por lo que estos enfoques solo son justificados cuando se obtiene un modelo con mejor desempeño al modelo base. En otros trabajos

se busca entrenar un modelo genérico para diferentes tareas [Rusu et al., 2016, Schmitt et al., 2018]. Sin embargo, en ocasiones el modelo olvida las tareas que fueron entrenadas al inicio (*catastrophic forgetting*), además de que en ocasiones se obtiene un modelo más complejo al inicial. Pocos trabajos abordan transferir hacia una nueva tarea, la mayoría usan técnicas como *fine-tuning* que consisten en utilizar una partes de redes neuronales artificiales para entrenar modelo a partir de los pesos iniciales [de la Cruz et al., 2016, Carr et al., 2018, Fawaz et al., 2018].

Basado en las problemáticas y factores presentados en los párrafos anteriores en esta tesis se plantea desarrollar una metodología para la transferencia de conocimiento utilizando múltiples tareas en algoritmos basados en DRL para reducir el tiempo de entrenamiento de una red neuronal artificial (*Artificial Neural Network, ANN*). La idea principal es que utilizando elementos relevantes de distintas CNN asociadas a las tareas base pueden reducir el tiempo de convergencia en agentes DRL.

Para ello se necesitan resolver algunos problemas como definir cuáles tareas base pueden ser útiles para aplicar la transferencia de aprendizaje. Para solventar este problema se propone una medida entre tareas basada en la comparación de espacios de acciones y la salida de una capa oculta de una CNN. La medida propuesta selecciona una tarea que mejora el desempeño respecto a la inicialización de pesos aleatorios.

2 Marco Teórico

En esta sección se presentan un resumen de los temas que conciernen a este trabajo de tesis. Primero se revisan los conceptos básicos de aprendizaje supervisado. En la siguiente sección se da una introducción sobre RL, que es uno de los temas centrales de este trabajo. Otro tema importante en esta tesis es el DL, del cual también se menciona en esta sección. Finalmente, se presenta un resumen sobre DRL que es la combinación entre RL y DL.

2.1 Aprendizaje Supervisado

El aprendizaje automático es una rama de la inteligencia artificial que se divide en tres campos: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo [Skansi, 2018].

El aprendizaje supervisado esta basado en aprender utilizando ejemplos etiquetados, las etiquetas clasifican los elementos en diferentes grupos y el objetivo es que el clasificador aprenda a distinguir entre estos grupos y puedan etiquetar ejemplos no etiquetados de manera correcta [Learned-Miller,].

Los algoritmos de clasificación más utilizados son los clasificadores lineales que se basan en el problema de clasificación, que consiste en separar puntos que pertenezcan a distintos grupos. Cada punto representa un individuo en el conjunto de datos y cada dimensión es una característica que describe a cada individuos. En la Figura 1 se muestra un ejemplo de un conjunto de datos donde se tienen dos grupos distintos con dos características, cada color representa un grupo de datos.

Para separar los datos se tiene que encontrar el hiperplano que separe de mejor manera los grupos basado en los datos disponibles. Una vez que se encuentra un hiperplano de separación se pueden clasificar nuevos datos, para el cual se encuentra de que lado del hiperplano se localiza el individuo a clasificar.

Cuando se pueden separar los grupos del conjunto de datos con un hiperplano recto se dice que los datos son linealmente separables como se muestra en la Figura 1. En dicha figura

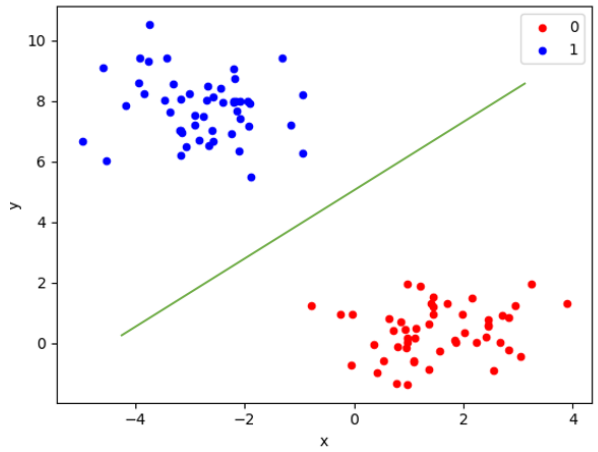


Figura 1: Ejemplo de un conjunto de datos con dos grupos, junto con un hiperplano de separación.

también se puede observar un hiperplano de separación del conjunto de datos. El número de hiperplanos que pueden separar los datos es infinito, el principal problema es encontrar uno que separe de mejor manera dichos datos.

Sin embargo, se puede encontrar el caso donde una simple línea o un hiperplano recto no es suficiente para separar los grupos del conjunto de datos. Un problema de datos que no son linealmente separables es la representación de la función lógica XOR, que se puede observar en la Figura 2. Para solventar estos problemas algunos algoritmos como Máquina de Vectores de Soporte (*Support Vector Machines*, SVM) [Cortes and Vapnik, 1995], añaden una dimensión más basados en las características que ya tiene o utilizan una transformación de los datos con un *kernel*.

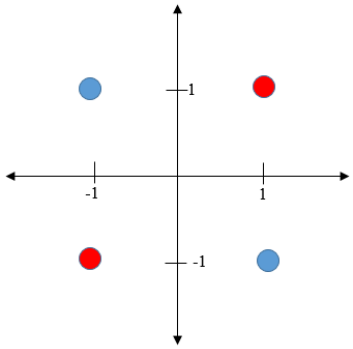


Figura 2: Ejemplo de datos que no son linealmente separables.

Los algoritmos de aprendizaje supervisado reciben como entrada un conjunto de datos de entrenamiento con sus respectivas etiquetas. En la primera parte se realiza el entrenamiento, donde se ajustan los hiperplanos utilizando el conjunto de entrenamiento y parámetros propios de cada algoritmo. En la siguiente fase se predicen los grupos a los que pertenecen nuevos datos, éstos son evaluados del algoritmo de aprendizaje supervisado utilizando el hiperplano que se ajustó en la etapa de entrenamiento.

Un algoritmo que ha tenido gran éxito para resolver el problema de clasificación son las redes neuronales. Este algoritmo es utilizado para aprendizaje supervisado, también son el algoritmo base para los enfoques basados en DL que es un tema principal de esta tesis. En las siguientes secciones se describe este algoritmo.

2.2 Redes Neuronales Artificiales

Las ANNs han demostrado ser un algoritmo de clasificación supervisado muy utilizado en los últimos años. Este clasificador esta inspirado en las conexiones que hay entre las neuronas del cerebro. Las ANNs constan de unidades conectadas entre sí, donde cada una tiene como entrada un conjunto de números reales y tienen como salida un número real que puede ser entrada de otra unidad [Mitchell et al., 1997]. Las unidades en las que están basadas las ANNs se llaman perceptrones y su representación gráfica se puede observar en la Figura 3.

Un perceptrón toma como entrada un vector de valores numéricos y calcula una combinación lineal de estás entradas como se muestra en la Ecuación 1, posteriormente se calcula la salida del perceptrón que es 1 si el resultado es mayor a un umbral (en el caso de la Figura 3 el umbral es igual a cero), y -1 en otro caso. Cada peso w_i determina la contribución de cada entrada x_i a la salida del perceptrón. El aprendizaje consiste en encontrar los mejores valores para los pesos $w_0, w_1, w_2, \ldots, w_n$.

$$o(x_1, x_2, \ldots, x_n) = \begin{cases} 1 \textbf{ si } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \textbf{ en otro caso} \end{cases} \tag{1}$$

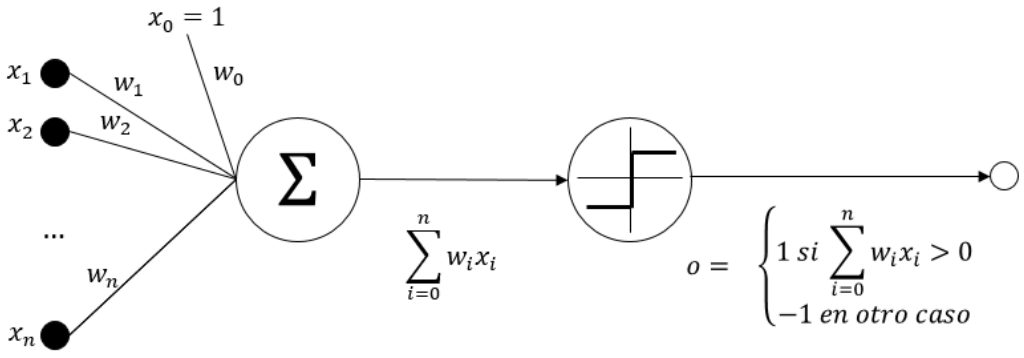


Figura 3: Representación gráfica de un perceptrón (imagen reproducida de [Mitchell et al., 1997]).

Para encontrar los pesos que produzcan la salida deseada para los ejemplos del conjunto de entrenamiento, se tienen que usar las Ecuaciones 2 y 3 (donde t es la salida esperada, o es la salida generada por el perceptrón y η es la tasa de aprendizaje que modera el grado en el que los pesos son modificados en cada paso), que son base para aprender ANNs con muchas unidades.

El proceso de entrenamiento comienza asignando valores aleatorios los pesos $(w_0, \ldots, w_n)$. Posteriormente, se modifican los pesos cada vez que los ejemplos se clasifiquen incorrectamente, este proceso se repite hasta que el perceptrón clasifique correctamente los ejemplos del conjunto de entrenamiento. Se debe tomar en cuenta que los perceptrones sirven para clasificar datos que son linealmente separables.

$$w_i \leftarrow w_i + \Delta w_i \tag{2}$$

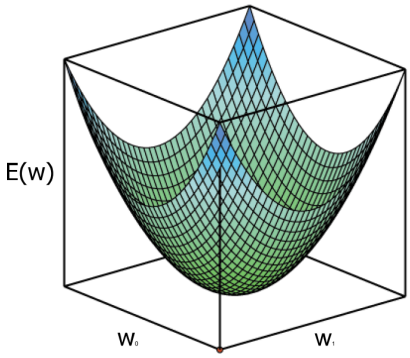


Figura 4: Espacio de soluciones.

donde,

$$\Delta w_i = \eta(t - o)x_i \tag{3}$$

Para encontrar los pesos que mejor separen los datos de entrenamiento se puede realizar una búsqueda mediante gradiente descendente, que ayuda a saber por en que regiones del espacio de soluciones llegar a un mínimo local. Por ejemplo en la Figura 4 se muestra el espacio de combinaciones entre dos pesos w_0 y w_1, mientras que en el otro eje se muestra el error, lo que se busca en este caso es llegar al error mínimo para el conjunto de entrenamiento.

En la actualización de pesos mediante gradiente descendente se tiene que definir la medida del error de la unidad. En este caso se utiliza el error cuadrático medio en la Ecuación 4, donde D es el conjunto de datos de entrenamiento, t_d y o_d son la clase a la que pertenece y la salida de la unidad del ejemplo d.

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \tag{4}$$

Al tener un conjunto de pesos asociados a las diferentes unidades dentro de la ANN, este se representa mediante un vector de pesos $\vec{w}$. Para actualizar los pesos extiende la Ecuación 3, utilizando la derivada de ésta, y se obtiene mediante la Ecuación 5, donde η determina que tanto se toma en cuenta la actualización del peso y es llamado tasa de aprendizaje [Mitchell et al., 1997]. Una variante del gradiente descendente es el gradiente descendente estocástico, donde se actualizan los pesos de manera gradual siguiendo el error de cada ejemplo.

$$\Delta w_i = \eta \sum_{d\in D}(t_d - o_d)x_{id} \tag{5}$$

Para encontrar hiperplanos que separen datos que no sean linealmente separables, se tiene que utilizar un tipo de unidad diferente al perceptrón. Una alternativa es la unidad sigmoide (ver Figura 5) que primero encuentra una combinación lineal de las entradas y los pesos para después procesar la salida mediante una función continua, que se muestra en la Ecuación 6.

$$\delta(y) = \frac{1}{1 + e^{-y}} \tag{6}$$

Como se mencionó anteriormente, los perceptones pueden separar datos que son separables por una línea recta. Para datos no linealmente separables se cambia la función de activación

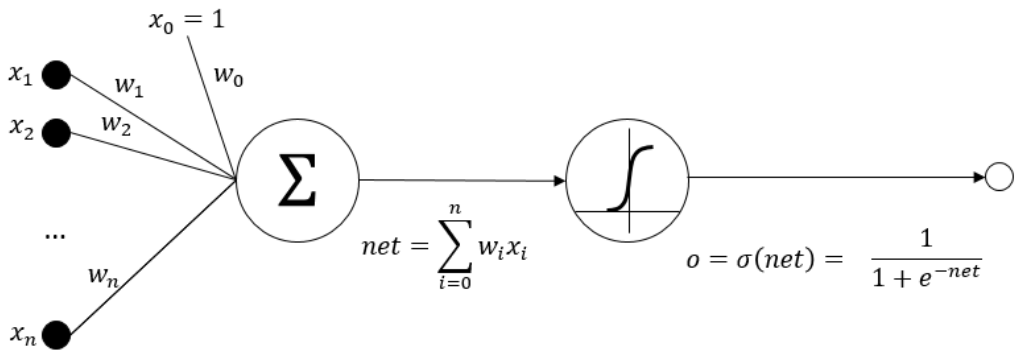


Figura 5: Representación gráfica de la unidad sigmoide (imagen reproducida de [Mitchell et al., 1997]).

diferente [1]. Utilizar un conjunto de unidades conectadas entre sí, también sirve para encontrar un hiperplano de separación de datos que no son linealmente separables.

Una ANN se puede representar como un grafo acíclico que conecta unidades para aprender un hiperplano de separación para el conjunto de entrenamiento. La arquitectura de una ANN consta de una capa de entradas, donde se dan como parámetros de entrada las características de los datos del conjunto de entrenamiento, una o más capas ocultas y una capa de salida con el número de clases a separar. Un ejemplo de una ANN con ocho unidades en la capa de entrada, tres unidades en su única capa oculta y ocho unidades en la capa de salida se muestra en la Figura 6.

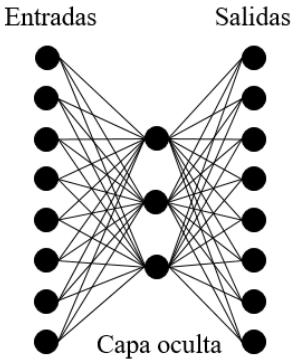


Figura 6: Ejemplo de una ANN.

Para aprender los pesos de una ANN se utiliza el algoritmo *backpropagation*, donde se busca minimizar el error utilizando gradiente descendente, este algoritmo garantiza encontrar un mínimo local. Primero, se tiene que definir una función de error que tome en cuenta cada unidad de salida en la ANN como se muestra en la Ecuación 7, donde *salidas* es el conjunto de unidades de salida, t_{kd} y o_{kd} son la salida deseada y la salida de la red respectivamente.

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in salidas} (t_{kd} - o_{kd})^2 \tag{7}$$

[1] La función de activación se refiere a la forma en que se procesan las entradas de la unidad después de realizar la combinación lineal entre los pesos y las entradas.

El algoritmo *backpropagation* se utiliza para entrenar una ANN, este se funciona de la siguiente manera. Se comienza construyendo la ANN e iniciar los pesos con números aleatorios pequeños. En el ciclo principal del algoritmo se itera sobre cada elemento del conjunto de entrenamiento y se calcula el error, después se encuentra el gradiente respecto al error y se actualizan los pesos de la red [Mitchell et al., 1997].

2.3 Aprendizaje Profundo

En el Aprendizaje profundo (DL, *Deep Learning*) se obtienen múltiples niveles de representación utilizando módulos no lineales, entre más se añaden se obtienen mayor nivel de abstracción [LeCun et al., 2015].

Una ventaja es que las características obtenidas son extraídas por el mismo clasificador y no por humanos, lo cual en algunos casos puede tener mejor desempeño [Goodfellow et al., 2016]. Las capas más profundas (con mayor nivel de abstracción) captura los aspectos más importantes para clasificar y suprime aquellos que son irrelevantes [LeCun et al., 2015].

Ya que se utilizan múltiples capas de ANNs, se tiene un conjunto de parámetros en el orden de millones, los cuales tienen que ser ajustados. Tomando en cuenta los anterior el proceso de entrenamiento puede ser muy tardado aún si se cuenta con *hardware* especializado.

Se han realizado distintos procesos para reducir el tiempo de entrenamiento como utilizar una función de activación que sea más fácil de obtener como la función ReLU (*Rectified Linear Unit*), que primero realiza una combinación lineal entre los pesos de entrada y se obtiene el máximo entre la salido o el valor cero como se muestra en la Figura 7. Además de lo anterior se ha utilizado *hardware* especializado como GPUs (Graphic Procession Units), que sirven para realizar operaciones en paralelo, inclusive en los últimos años se ha desarrollado hardware especializado para operaciones entre tensores (*Tensor Processing Units*) [TPU,].

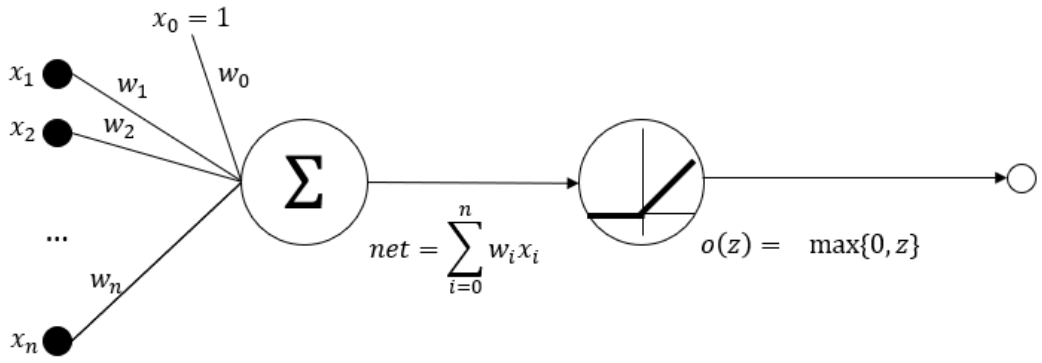


Figura 7: Función de activación ReLU.

Otra limitación es que las redes neuronales profundas necesitan un gran número de instancias para tener un desempeño aceptable, para lidiar con este problema se han realizado distintas tareas como rotar las imágenes de entrada para tener más instancias y que la clasificación sea más robusta como se hace en [Krizhevsky et al., 2012].

Las redes neuronales convolucionales se han convertido en uno de los algoritmos de clasificación supervisada más utilizado de debido a que se puede tener como entrada datos en bruto (sin procesar) como: imágenes, señales, entre otras. En la siguiente sección se resume su funcionamiento.

2.3.1 Redes Neuronales Convolucionales

Las redes neuronales convolucionales (*Convolutional Neural Networks*, CNN) son un tipo de redes neuronales que se utilizan para datos en bruto, que siguen una estructura de cuadrícula, la cual puede tener múltiples dimensiones. Ejemplos de estos pueden ser series de tiempo (con una dimensión) o imágenes que pueden tener dos (imágenes en escala de grises) o tres (imágenes con tres canales, por ejemplo RGB) dimensiones respectivamente. Su nombre se lo deben a que utilizan la operación matemática llamada convolución.

Para aplicar la convolución se utilizan dos argumentos. El primero es la entrada, que son los datos en bruto I y el *kernel* K, acorde a la Ecuación 8. En la Figura 8 se puede ver de manera gráfica como se aplica la convolución, similar al proceso de filtros regionales imágenes digitales [Burger et al., 2009]. La convolución consta en aplicar un barrido por todo el dato de entrada con el *kernel* y se obtiene una nueva imagen. Para el proceso se toma en cuenta el incremento para realizar el barrido del *kernel* en los datos de entrada[2].

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{8}$$

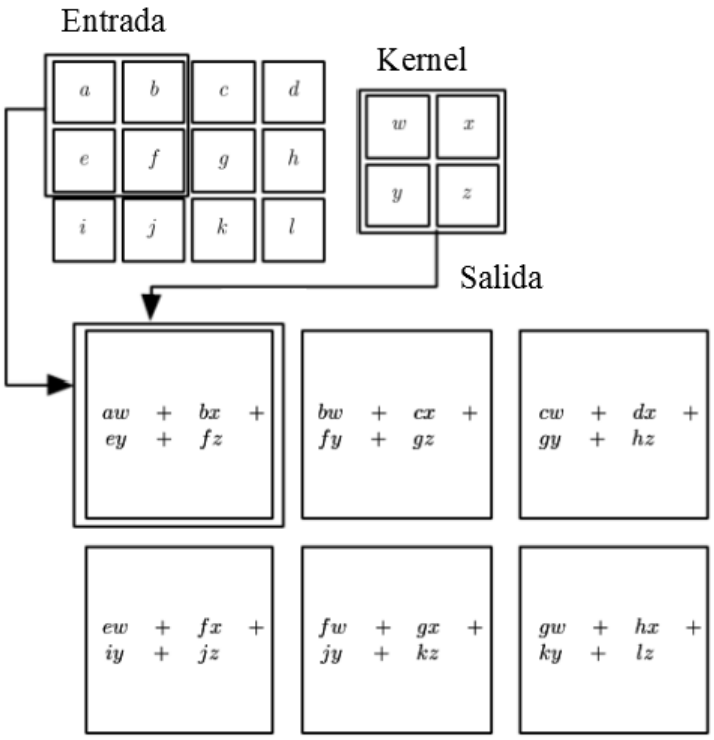


Figura 8: Proceso de convolución, imagen tomada de [Goodfellow et al., 2016].

Una capa de convolución consiste en tres partes. Primero, se realiza una serie de operaciones convolucionales con una serie de *kernels*, lo que produce imágenes con diferente número de canales. En la segunda fase, se aplica la función activación, por lo regular de tipo ReLU (Figura 7). En la última fase se aplica una función que resume una región en los datos de entrada, esta función es llamada *pooling*. Algunos ejemplos de las funciones *pooling* puede

[2]El incremento tiene que ver con cuantas unidades se mueve el *kernel*

ser el valor máximo, el promedio o el mínimo de una región. Las tres etapa se muestran gráficamente en la Figura 9.

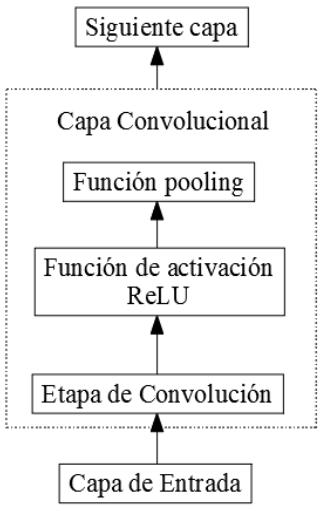


Figura 9: Capa de Convolución, imagen reproducida de [Goodfellow et al., 2016].

Las tres etapas anteriores se repiten en diferentes capas convolucionales. Las capas más profundas capturan características más específicas del conjunto de entrenamiento, mientras que las primeras capas capturan características más generales [Yosinski et al., 2014].

Después de las capas de convolución se realiza una operación *flatten* donde se utilizan como características los valores de las imágenes resultantes de aplicar las capas de convolución. En las siguientes capas se utilizan las capas tradicionales de una ANN (llamadas completamente conectadas). Finalmente en la capa de salida se dan las clases que se quieren clasificar.

El proceso de diseño de una red neuronal convolucional tiene diferentes parámetros a tomar en cuenta. Para las capas convolucionales se tiene que establecer el número y tamaño de los *kernels*, así como la función de *pooling* a utilizar y el incremento para aplicar la convolución. En las capas completamente conectadas se tiene que establecer el número de neuronas y la función de activación. Finalmente se tienen que tomar en cuenta los parámetros propios de las ANNs como la tasa de aprendizaje, momentum, etc.

Una limitación del DL es que necesita una gran cantidad de datos para alcanzar un desempeño aceptable, este es un gran problema debido a que no siempre se tienen los datos para entrenar una CNN con millones de parámetros. Para solventar este problema se han propuesto diferentes soluciones como aumentar el número de ejemplos en el conjunto de entrenamiento realizando distintas transformaciones a las imágenes como se muestra en [Krizhevsky et al., 2012]. Otra manera de solventar este problema es utilizarla técnica *dropout*, donde se toman en cuenta solo un porcentaje de unidades para el entrenamiento, esto reduce el sobre ajuste en la red neuronal [Srivastava et al., 2014].

2.4 Aprendizaje por Refuerzo

Uno de los objetivos de la inteligencia artificial es crear agentes inteligentes que aprendan mediante la interacción. El aprendizaje por refuerzo, es un enfoque que se basa en la premisa anterior, este se puede incluir como un tercer paradigma de aprendizaje computacional junto con el aprendizaje supervisado y el aprendizaje no supervisado [Sutton, 2018].

En RL, un agente interactúa mediante una representación del ambiente, a la cual se le llama estado, que muestra al agente como es que se encuentra el ambiente, para realizar una acción. El objetivo del agente es maximizar la recompensa obtenida.

Además del agente y el ambiente se pueden mencionar cuatro elementos del aprendizaje por refuerzo: una política que determina como se comporta el agente en un estado; una función de recompensa que puede ser positiva o negativa dependiendo de la decisión que tome el agente; una función de valor que especifica el valor total de recompensa que se espera obtener a largo plazo comenzando en un estado; de manera opcional un modelo que imita el comportamiento del ambiente.

Un proceso de decisión de Markov (*Markov Decision Process*, MDP) es una formalización de un problema de toma de decisiones secuencial. Donde se toman en cuenta recompensas futuras. Un MDP es considerado como una abstracción de un problema donde se busca alcanzar una meta mediante la interacción entre un agente y el ambiente.

Estos dos elementos (ambiente y agente) interactúan en una secuencia discreta de pasos $t = 0, 1, 2, 3, \ldots$. En cada paso t, el agente recibe una representación del ambiente (estado), $S_t \in S$ y basado en esto el agente realiza una acción $A_t \in A(s)$. Tomando en cuenta qué tan buena o mala fue la decisión que el agente tomó, éste recibe una recompensa numérica, $R_{t+1} \in \mathcal{R} \in \mathbb{R}$ y se encuentra el siguiente estado S_{t+1}.

La interacción descrita anteriormente se puede ver gráficamente en la Figura 10. El proceso descrito anteriormente es repetido y se encuentra una secuencia como en la Ecuación 9.

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \tag{9}$$

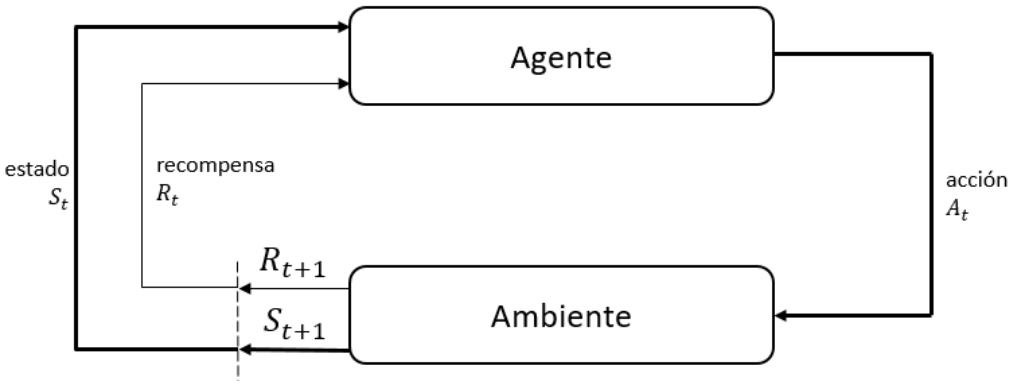


Figura 10: Trayectoria que sigue un MDP, interacción entre el agente y el ambiente [Sutton, 2018].

De manera simplificada un MDP se define como una quintupla $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$:

- $\mathcal{S}$, un conjunto de estados.
- $\mathcal{A}$, que es un conjunto de acciones disponibles para el agente, tomando en cuenta el estado actual.
- Una función de recompensa $\mathcal{R}(\mathcal{A}, \mathcal{S})$, que retorna un valor numérico dependiendo la acción tomada por el agente.
- Una función de transición que define la probabilidad de pasar a un estado s' estando en un estado s, tomando una acción a, $P(s'|s, a)$.

- Un factor de descuento γ que ayuda a tomar en cuenta en menor medida recompensas futuras mediante la ecuación de Bellman.

La meta de una agente es maximizar el retorno esperado G_t, que esta definido como una función específica de la secuencia de recompensas como en la Ecuación 10. Donde T es el último paso (estado final) de la secuencia, la cual es llamada episodio.

Como se mencionó anteriormente, el agente busca maximizar la recompensa obtenida en el episodio, acorde a la Ecuación 10 las recompensas futuras son tomadas en cuenta de igual manera que la recompensa actual. Sin embargo, recompensas futuras pueden tener menos impacto sobre estados más cercanos, por lo que se usa el factor γ que sirve para tomar en cuenta de menor manera recompensas futuras como se muestra en la Ecuación 11.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{10}$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{11}$$

Los algoritmos de RL se busca encontrar funciones de valor, donde se estima qué tan bueno es para un agente estar en un estado, en términos de recompensa futuras. Basado en las funciones de valor se puede encontrar una política π, la cual sirve para determinar cómo el agente se va a comportar de acuerdo a los estados proporcionados. Formalmente, una política es un mapeo de estados a probabilidades de tomar una acción disponible.

Para encontrar una política se pueden calcular dos funciones, la función de valor $v(s)$ donde se estima la recompensa esperada comenzando en el estado s siguiendo una política π (Ecuación 12). De manera similar, se puede estimar la función acción-valor $q_\pi(s, a)$ que estima la recompensa esperada comenzando en un estado s tomando una acción a y después siguiendo la política π (Ecuación 13).

$$v_\pi = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \forall s \in \mathcal{S} \tag{12}$$

$$q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \forall s \in \mathcal{S} \tag{13}$$

Un agente tiene un conjunto de políticas que son evaluadas de acuerdo a la recompensa esperada obtenida para todos los estados. Es decir, $\pi \geq \pi'$ si y solo si $v_\pi \geq v_{\pi'}$. Al menos una política obtiene mayor recompensa esperada, ésta es la mejor y es llamada política óptima π_*. De manera similar se puede encontrar las funciones de valor v_* y de acción-valor q_* óptimas.

Si se tiene un espacio pequeño de estados-acciones es fácil encontrar una política óptima utilizando una tabla con las posibles combinaciones entre los estados y las acciones. Sin embargo, en la práctica no siempre se encuentran las políticas óptimas, ya que se tiene un espacio grande de estados-acciones. Lo cual en algunos casos, representarlos en forma de tabla es un problema intratable. En estos casos se encuentra una aproximación de la política óptima, por ejemplo utilizando clasificadores, como lo hacen los algoritmos basados en gradientes.

Para encontrar la política óptima, primero se tienen que evaluar la función de valor para cada estado (evaluación de la política) y posteriormente evaluar que tan buena es la nueva política π' respecto a la anterior π basado en la recompensa obtenida, de tal manera que si $\pi' \geq \pi$ se actualiza la política y se queda con la mejor.

Considere el ejemplo de la Figura 11, donde se aplica un enfoque basado en programación dinámica, este tipo de enfoques se utilizan cuando se tiene un modelo completo del ambiente. En el ejemplo se tiene un conjunto de cuatro acciones $\mathcal{A} = \{\mathsf{arriba}, \mathsf{abajo}, \mathsf{izquierda}, \mathsf{derecha}\}$, la meta es encontrar el camino más corto a los estados finales marcados en la figura como cuadros sólidos, la política inicial es completamente aleatoria, la recompensa es -1 en todos los pasos excepto en los estados terminales donde es 0.

En el ejemplo mencionado anteriormente, se comienza con una política aleatoria y el mismo valor para todos los estados $k = 0$. Estos valores se modificaran de acuerdo a la Ecuación 14, excepto los estados finales que permanecen siempre con un valor de cero. En el siguiente paso se realiza una actualización de v para todos los estados $k = 1$ y se actualiza la política. En este paso solo los valores de los estados vecinos a los finales son cambiados y se obtiene una política mejor ya que en éstos se llaga con una probabilidad más alta a los estados finales. Este proceso (*Generalized Policy Iteration*, GPI) es repetido hasta encontrar la política óptima, que es donde se obtiene la recompensa máxima.

$$v(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{14}$$

2.5 Aprendizaje por Refuerzo Profundo

Para los casos anteriores se considera un espacio de estados-acciones pequeño, donde estas combinaciones se pueden representar en forma de tabla. Sin embargo, existen casos donde no es posible representar estas combinaciones como tabla debido a que dicho espacio es muy grande.

Considere el ejemplo de la Figura 12, donde cada punto es un estado del juego *Space Invaders*. En dicha imagen se muestra una proyección de la salida de la última capa oculta de una *Deep Q-Network* (DQN) entrenada para este juego, utilizando el algoritmo t-SNE. Este es una de los casos donde representar dichos estados es muy difícil debido a la que se necesita muchos recursos computacionales para hacerlo, debido a la dimensión de los estados y a las posibles combinaciones de los éstos y acciones. Para estos casos se utiliza un algoritmo de clasificación supervisada para predecir los valores de v, q, o ambos dependiendo el enfoque utilizado.

Debido a las ventajas que tiene DL (i.e. se extraen de manera automática características, parte de datos en bruto), este tipo de enfoques se ha combinado con RL (DRL) para entrenar agentes inteligentes. Estos algoritmos ajustan un conjunto de parámetros $\theta \in \mathbb{R}^d$ para encontrar una política $\pi(a|s, \theta) = PrA_t = a|S_t = s, \theta_t = \theta$, donde se encuentran las probabilidades de utilizar una acción a en un estados s con los parámetros θ.

Al igual que los enfoques mencionados anteriormente, se busca mejorar el maximizar la recompensa obtenida mediante actualizaciones de θ utilizando una medida de desempeño, que es un escalar $J(\theta)$ usando gradiente ascendente (Ecuación 15, donde $\widehat{\bigtriangledown J(\theta_t)}$ es la estimación del gradiente ascendente y α es la tasa de aprendizaje).

Estos métodos de tipo Monte Carlo ya que se basan únicamente episodios sobre una política aleatoria o tipo ϵ-*greedy* donde el valor ϵ determina con qué probabilidad se utilizaran acciones aleatorias. Este valor puede comenzar como un valor grande y se va reduciendo a medida que se van generando los episodios. A partir de estas simulaciones se actualizan los parámetros θ.

$$\theta_{t+1} = \alpha \widehat{\bigtriangledown J(\theta_t)} \tag{15}$$

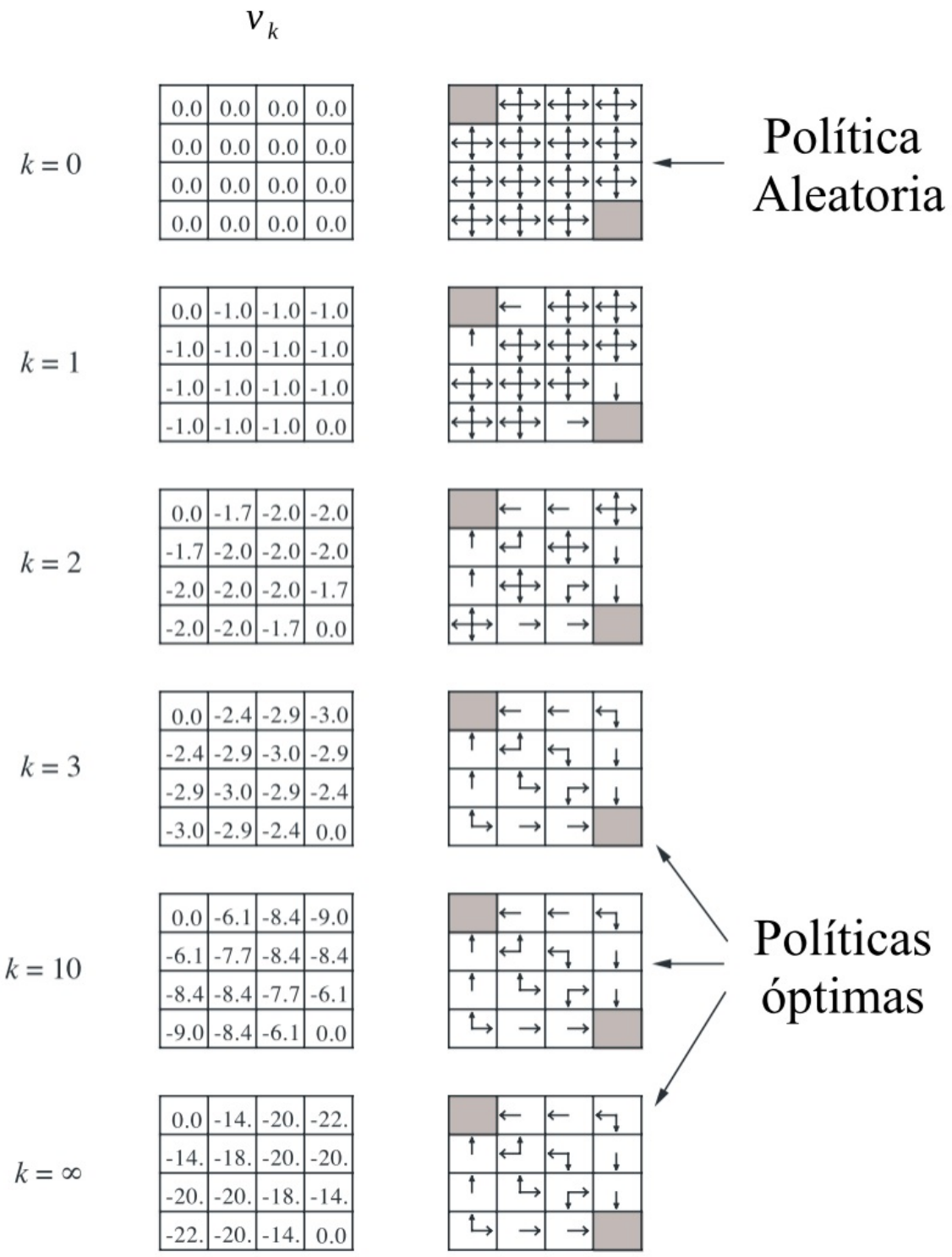


Figura 11: Ejemplo del proceso iterativo para encontrar la política óptima utilizando un enfoque de programación dinámica, imagen traducida de [Sutton, 2018].

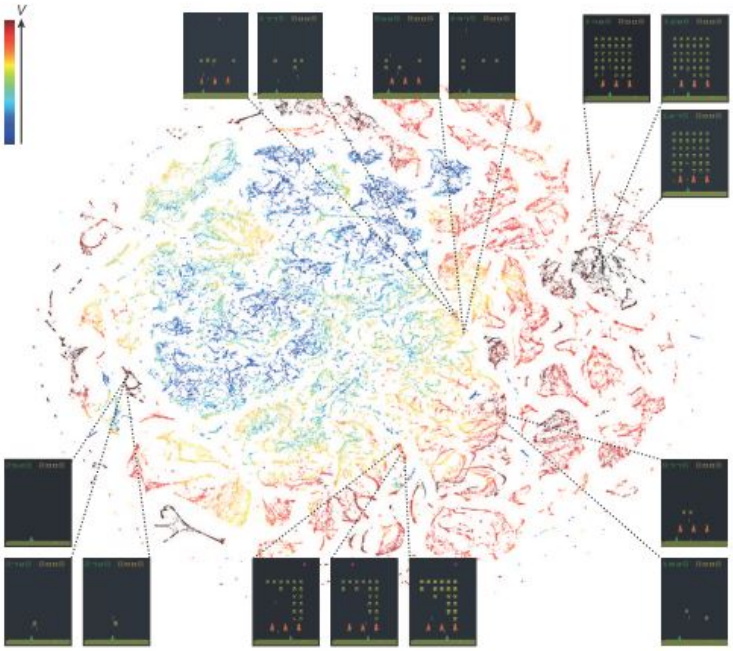


Figura 12: Ejemplo espacio de estados con su función de valor, cada punto representa un estado del juego *Space Invaders*. Dicha imagen corresponde a una proyección con el algoritmo t-SNE utilizando la salida de la última capa oculta de una DQN. Imagen de [Mnih et al., 2015].

En estos métodos por lo regular se utiliza una ANN profunda o una CNN, donde los parámetros θ son los pesos de la red neuronal utilizada y las salidas corresponden a las acciones disponibles en el MDP. Una desventaja de estos enfoques es que pequeños cambios en θ pueden cambiar mucho la política.

Estos métodos aproximan alguno de los elementos del MDP como la función de valor (algoritmo REINFORCE [Sutton, 2018]), la función q (*Deep Q Network* (DQN) [Mnih et al., 2015]) o ambas (*actor-critic* [Mnih et al., 2016]).

El algoritmo que se tomará como base para esta propuesta es donde se aproxima el valor q [Mnih et al., 2015], debido a que este fue el primer enfoque que obtuvo buenos resultados, en muchos de ellos se mejora el desempeño respecto a puntuaciones obtenidas por humanos en el *baseline* de juegos de Atari. En el se propone utilizar una CNN para aproximar el valor q, que es la recompensa a largo plazo generada en un episodio. Este algoritmo resultó ser robusto al obtener buenos resultados para 49 diferentes juegos de Atari.

En los últimos años estos enfoques han tenido grandes avances debido a que se tienen computadoras más potentes, lo que hace menos pesado el entrenamiento de CNN. En la Sección 3.1 se presentan algunos enfoques que muestran como han evolucionado.

2.6 Transferencia de Conocimiento

La transferencia de conocimiento (*Transfer Learning*, TL) es una técnica utilizada cuando en la tarea objetivo no se tiene un conjunto de datos suficiente para alcanzar un desempeño aceptable. Lo que se esperaría obtener al aplicar técnicas de TL se puede observar gráficamente en la Figura 13. Donde se muestran dos curvas de aprendizaje, una con un entrenamiento sin

transferencia (línea punteada) y otra realizando transferencia (curva sólida). Lo que se busca es obtener el mismo desempeño en menor tiempo, por ejemplo en el punto de evaluación en la gráfica. Una métrica importante es el *jumpstart* que es la ganancia de desempeño en el primer paso de entrenamiento, mientras que la ganancia de desempeño al final del entrenamiento.

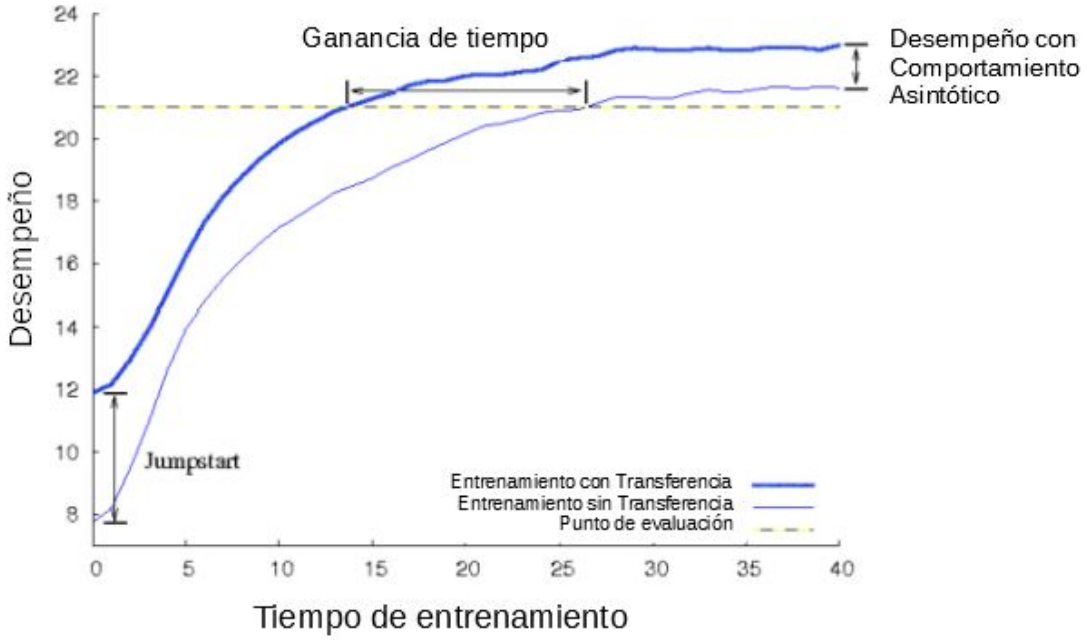


Figura 13: Gráfica sobre lo que se espera obtener al realizar transferencia de conocimiento (imagen reproducida de [Taylor and Stone, 2009]).

En las técnicas basadas en TL se tienen dos tareas las cuales denominadas como tarea base y tarea objetivo. La primera de ellas tiene un conjunto de datos etiquetados suficiente para encontrar un modelo que clasifique los datos de manera aceptable. Sin embargo, en la tarea objetivo se tienen pocos ejemplos etiquetados y de manera opcional se tiene un conjunto de datos no etiquetados. La transferencia de aprendizaje consiste en utilizar el modelo entrenado para la tarea base para acelerar o mejorar el entrenamiento de la tarea objetivo. Siguiendo con la notación de [Wang and Deng, 2018] a continuación se presentan algunas definiciones relevantes sobre transferencia de conocimiento.

Un dominio $\mathcal{D}$ consiste en un espacio de características $\mathcal{X}$ y una distribución de probabilidad marginal $P(X)$. Donde $X = \{x_1, x_2, x_3, x_3, \dots\}$, donde el valor x_i esta dado por un vector $d-$dimensional con valores numéricos. Mientras que una tarea $\mathcal{T}$ consiste en un espacio de características $\mathcal{Y}$, donde $Y = y_1, y_2, y_3, \dots$, que corresponden a los grupos que existen en la $\mathcal{T}$ y una función de predicción $f(\cdot)$, la cual se puede ver como una distribución de probabilidad condicional $P(X|Y)$. Los elementos del conjunto etiquetado se ordenan en pares (x_i, y_i), donde $x_i \in \mathcal{X}$ y $y_i \in \mathcal{Y}$ [Weiss et al., 2016].

La adaptación de dominio (DA, *Domain Adaptation*) es un caso específico de TL, el cual es utilizado en este proyecto. En DA se utilizan datos etiquetados de uno o más dominios base para ejecutar una tarea objetivo [Kouw, 2018]. En DA se asume que se tienen dos dominios: un conjunto de entrenamiento con suficientes datos es un dominio base $\mathcal{D}^S = \{\mathcal{X}^S, P(X)^S\}$, y un conjunto con pocos datos etiquetados o no es un dominio objetivo $\mathcal{D}^T = \{\mathcal{X}^T, P(X)^T\}$.

Como se mencionó anteriormente en el $\mathcal{D}^T$ se puede tener una parte del conjunto de

datos etiquetada $\mathcal{D}^{tl}$ y una parte no etiquetada D^{tu}, de manera que el dominio objetivo es la unión entre los dos últimos $\mathcal{D}^T = \mathcal{D}^{tl} \cup \mathcal{D}^{tu}$. Cada dominio tiene asociada una tarea $\mathcal{T}^S = \{\mathcal{Y}^S, P(Y^S|X^S)\}$ para el dominio base y para la tarea objetivo $\mathcal{T}^T = \{\mathcal{T}^S, P(Y^T|X^T)\}$. De manera similar, $P(Y^S|X^S)$ se puede aprender del conjunto de datos base (el cual esta etiquetado) $\{x_i^s, y_i^s\}$, mientras que se puede aprender con la parte de etiquetada del conjunto objetivo $\{x_i^{tl}, y_i^{tl}\}$ y la parte no etiquetada $\{x_i^{tu}\}$.

La transferencia de aprendizaje se puede definir como el proceso de mejora en la función de predicción en el dominio objetivo $f_t(\cdot)$ usando la información del dominio base. El TL, la divergencia entre conjuntos de datos puede ser causada por diferencias entre dominios $\mathcal{D}^{\mathcal{S}} \neq \mathcal{D}^{\mathcal{T}}$, divergencia entre tareas $\mathcal{T}^S \neq \mathcal{T}^T$ o ambos. Como se mencionó anteriormente, DA es un caso específico de TL. Formalmente, la tarea base y la tarea dominio es la misma $\mathcal{T}^S = \mathcal{T}^T$ y los dominios son diferentes $\mathcal{D}^S \neq \mathcal{D}^T$.

En TL sobre ANN, se utiliza una técnica llamada *fine-tuning*, que consiste en reutilizar los pesos (o los pesos de algunas capas) de una ANN que fue entrenada en un dominio (tarea base) y tiene un buen desempeño en dicha tarea, para entrenar el modelo asociado a la tarea objetivo con dichos pesos. Se pueden definir dos esquemas para realizar el *fine-tuning*. En un esquema se dejan fijas los pesos de algunas capas de la ANN y solo se entrenan las últimas; mientras que en otro se entrena toda la red como se muestra en [Yosinski et al., 2014].

La limitación más grande de TL es la transferencia negativa que es cuando al entrenar la ANN en un nuevo dominio se obtiene un desempeño peor comparado con un entrenamiento al iniciar el modelo con pesos aleatorios. Para evitar estos problemas se ha propuesto realizar distintas configuraciones en los parámetros de entrenamiento como la iniciar de los pesos en la red neuronal o reducir la tasa de aprendizaje [Fawaz et al., 2018].

3 Estado del Arte

En esta sección se presenta el estado del arte sobre dos temas referentes a este proyecto de tesis. Primero se presentan los trabajos relacionados al DRL, debido a que son los algoritmos base para realizar transferencia de conocimiento hacia nuevas tareas. Posteriormente, se presentan los trabajos relacionados a transferencia de conocimiento en DRL.

3.1 Aprendizaje por Refuerzo Profundo

El DRL es un tema de interés debido a que en estos enfoques combinan las ventajas de DL, principalmente la abstracción de las instancias de entrada para resolver problemas utilizando RL. Una tarea compleja que se ha convertido en *baseline* para estos enfoques es aprender a jugar vídeo juegos de la consola Atari [Bellemare et al., 2013]. La mayoría de esos trabajos utilizan como entrada imágenes de los juegos, el espacio de acciones disponible (movimientos del *joystick* y combinaciones con un botón) y la función de recompensa utilizado una CNN para encontrar una política óptima.

Los trabajos relacionados a DRL se pueden dividir en tres grupos dependiendo de como se observa la salida, algunos realizan una estimación del valor Q [Mnih et al., 2015, van Hasselt et al., 2015], estos enfoques fueron los primeros en aparecer y su principal desventaja es que utilizan *experience replay*, el cual es una estructura de datos, donde se guardan instancias para realizar el entrenamiento, algunos algoritmos eligen los elementos de manera aleatoria y otros se basan en el error, la desventaja es que se utiliza mucha memoria para guardar la estructura (para entrenar el algoritmo DQN, se utiliza una *experience replay* de 1 millón de elementos, el cual utiliza aproximadamente 10 GB de memoria).

El segundo grupo se basa en estimar la política y la función Q [Mnih et al., 2016, Wang et al., 2015], en estos enfoques se deja de utilizar *experience replay*, su principal desventaja es el ajuste de parámetros en cada juego. Finalmente, se tiene un tercer grupo que propone ver la salida como una distribución de probabilidad [Hessel et al., 2017, Bellemare et al., 2017, Dabney et al., 2018a], estos se han convertido en el estado del arte para el *baseline* de juegos de Atari. En los siguientes párrafos se presentan los trabajos más relevantes y en la Tabla 1 se presenta un resumen de dichos trabajos con sus características.

La primera aproximación donde se propone la combinación entre DL y RL se propuso en 2013 [Mnih et al., 2013], este enfoque utiliza una CNN para predecir el valor Q (DQN). Posteriormente este trabajo fue extendido [Mnih et al., 2015] demostrando que este es un algoritmo robusto ya que se utiliza la misma arquitectura de red neuronal para aprender a jugar diferentes vídeo juegos, superando el desempeño de jugadores expertos en algunos casos.

Dos factores importantes para el desempeño de DQN: *experience replay* que consiste en almacenar un conjunto de instancias para entrenar la red, eligiendo de manera aleatoria con una distribución uniforme instancias de dicho conjunto para optimizar los pesos de la ANN. El otro es utilizar una copia de la red neuronal (*target network*) que se actualiza cada C pasos, esta se utiliza para generar episodios. En la red original de la cual se actualizan los pesos en cada iteración.

DQN tiene el problema de sobre estimación del valor Q (se sobre estima el valor de salida en algunas acciones, lo que se traduce en tener una política con mucha variación en la recompensa). *Double q-learning* (DDQN) ayuda a reducir la sobre estimación del valor Q. Este algoritmo consiste en tener dos estimaciones del valor Q, una para elegir acciones aleatorias mediante una política ϵ-*greedy* (*online network*[3]) y otra para elegir la mejor acción según la estimación (target network), después de algún número de episodios las dos estimaciones son cambiadas [van Hasselt et al., 2015].

Experience replay utiliza mucha memoria, lo cual es una limitación si no se cuenta con equipo especializado (10 GB para 1,000,000 de instancias que se es el valor que se da en [Mnih et al., 2015]). En [Schaul et al., 2015] se propone un método que da prioridad a las instancias que contribuyen más con el entrenamiento utilizando *TD-error*, de esta manera se mejora el desempeño respecto a DDQN, este método de selección de instancias se denomina *prioritized experience replay*.

Tomando en cuenta que no es necesario elegir todas las acciones en un determinado estado, [Wang et al., 2015] propone una arquitectura de CNN con dos partes: una parte determina cuales acciones son las mejores para un determinado estado y la otra parte predice el valor Q para cada accion. Esto mejora el desempeño de algunos juegos de Atari respecto a DQN [Mnih et al., 2015] y a DDQN [van Hasselt et al., 2015], se sigue utilizando *experience replay*.

En [Mnih et al., 2016] se propone dejar de usar *experience replay*, además de entrenar el agente en un CPU (no requiere tanto poder de cómputo). Este algoritmo se basa en utilizar múltiples copias del agente con diferentes configuraciones para la política exploratoria y se guarda un conjunto de instancias para ajustar los pesos de la red. Se utilizan diferentes enfoques para producir los episodios como *actor-critic* o SARSA.

Existen métodos que en vez de estimar un solo valor de salida para cada acción se aprende una distribución de probabilidad, Rainbow [Hessel et al., 2017] es el actual estado del arte y combina muchas de las mejoras en DQN, se utiliza *prioritized experience replay*, una arquitectura tipo *dueling*, actualizaciones utilizando n-pasos, y predice una distribución por cada acción disponible como lo hace C51. El citado trabajo [Bellemare et al., 2017] y como

[3]Esta es la red que se actualiza después de cada iteración en DQN

ya se mencionó se predice una distribución de probabilidad para cada acción basada en la recompensa obtenida. Finalmente, [Dabney et al., 2018a] propone una arquitectura donde se encuentra una distribución de cuantiles para cada acción, este enfoque mejora el desempeño de algunos juegos respecto a C51.

3.2 Transferencia en Aprendizaje por Refuerzo Profundo

En general existen tres esquemas de transferencia de conocimiento en DRL: una es realizar transferencia hacia una nueva tarea utilizando los pesos de una CNN pre-entrenada; otro es entrenar un modelo para dos o más tareas y el último es *policy distillation* que consiste en entrenar un modelo reducido para la misma tarea y que tenga un desempeño semejante o mejor al primero.

Los trabajos relacionados sobre los métodos mencionados anteriormente se presentan en la Tabla 2. En dicha tabla muestra el método utilizado para la transferencia, ya que la mayoría de los trabajos se basan en *policy distillation* se muestra una columna donde se puede ver si el método mencionado esta basado en *distillation*, posteriormente el esquema de transferencia que puede ser entre una tarea base y una objetivo o de múltiples tareas hacia una o entre un modelo genérico para múltiples tareas, después se describe como es el modelo resultante y finalmente se muestra un resumen sobre los aportes y limitaciones.

Los métodos de *policy distillation* [Parisotto et al., 2016, Schmitt et al., 2018, Rusu et al., 2015, Yin and Pan, 2017, Rusu et al., 2016, Zhuang et al., 2015] consisten en utilizar un modelo pre-entrenado para entrenar otro modelo con menos unidades en su arquitectura, lo que significa tener menos parámetros que optimizar, por lo que se necesitan menos instancias para alcanzar un desempeño igual o mejor al modelo pre-entrenado. Algunos trabajos han realizado mejora a la función de perdida o utilizan trazas con el agente base o con el modelo objetivo para entrenar un nuevo modelo. La principal desventaja es que no siempre se obtiene un mejor desempeño en el nuevo modelo, además de que la respuesta al evaluar una ANN no es computacionalmente costoso, por lo que la implementación de este tipo de métodos solo es justificable si se alcanza un desempeño mejor que el modelo pre-entrenado, además de que en la mayoría de los casos no sirve para realizar transferencia hacia una nueva tarea.

Otro tipo de enfoques se basan en reutilizar los pesos de un modelo pre-entrenado para entrenar uno nuevo en una tarea diferente (basados en *fine-tuning*), los métodos han mostrado ser efectivos en la reducción del tiempo de entrenamiento [Mittel et al., 2018, de la Cruz et al., 2016, Fawaz et al., 2018]. Sin embargo, los experimentos se han hecho suponiendo que las tareas (juegos de Atari en la mayoría) son parecidos, es decir no existe una manera numérica de evaluar qué tan distintas son las tareas o la manera de seleccionarlas, además de que pocos trabajos han abordado este tema.

El método más cercano a la medida propuesta se presenta en [Fawaz et al., 2018], donde se propone una métrica entre diferentes dominios, la cual esta basada en una reducción de dimensiones del conjunto de entrenamiento y se realiza *fine-tuning* para entrenar un modelo en el nuevo dominio. Este enfoque no se utiliza para DRL, sino para DL. En algunos casos la métrica no funciona para realizar la transferencia de aprendizaje.

Un último método utiliza la ventaja de encontrar un representación intermedia con una ANN (*Autoencoders* [Carr et al., 2018] o modelos generativos [Mittel et al., 2018]). Una vez que se encontró esa representación intermedia se entrena una parte del modelo para obtener una política. La principal desventaja es que se tiene que entrenar más de un modelo, lo que en la practica puede ser más tardado que entrenar un modelo con una DQN.

A diferencia de los trabajos presentados en esta sección, este proyecto de tesis se centra en

realizar transferencia de aprendizaje hacia una nueva tarea, además de usar múltiples tareas para transferir conocimiento a una nueva tarea. En los trabajos donde se realiza transferencia a una nueva tarea se menciona que las tareas se seleccionan de manera intuitiva basado en similitud de la forma de jugar. En este trabajo se plantea utilizar una medida entre tareas, esto con el objetivo de tener una forma más robusta para seleccionar las tareas para posteriormente transferir elementos relevantes hacia la tarea objetivo.

Tabla 1: Trabajos relacionados a DRL.

Algoritmo	**Experience Replay**	**Optimizer**	**Target Network**	**Arquitectura de ANN**	**Aportes**	**Limitaciones**
[Mnih et al., 2013] (DQN)	Uniforme	RMSProp	Si	CNN	Se obtuvieron buenos resultados en 7 juegos de Atari utilizando la misma arquitectura de CNN y mismos parámetros	Sobre estimación del valor Q. *Experience replay* usa mucha memoria, pequeños cambios en los pesos modifican significativamente la política
[Mnih et al., 2015] (DQN)	Uniforme	RMSProp	Si	CNN	Se extienden los experimentos de [Mnih et al., 2013], se usan 47 juegos. En algunos juegos se obtienen mejores resultados que humanos expertos	Sobre estimación del valor Q. *Experience replay* usa mucha memoria, pequeños cambios de los pesos modifican significativamente la política
[van Hasselt et al., 2015] (double DQN)	Uniforme	RMSProp	Si	CNN	Se utiliza la *target network*, para estimar los valores de salida en vez de números aleatorios, esto ayuda a realizar un mejor muestreo respeco a DQN	Se sigue teniendo el problema de *experience replay*, donde se necesita mucho memoria para guardar las instancias.
[Schaul et al., 2015] (Prioritized experience replay)	*Prioritized experience replay*	RMSProp	Si	CNN	Se da un peso a las transiciones que tengan un mayor TD-error, usa el mismo esquema que *double DQN*, se mejora en casi todos los juegos respecto a *double DQN*	Se guardan transiciones que tal vez no sean usadas, se reduce la exploración aunque se obtienen buenos resultados, se utiliza el mismo número de ejemplos en el *experience replay*
[Wang et al., 2015] (Dueling network)	*Prioritized experience replay*	RMSProp	Si	Tipo *dueling*	Se propone una arquitectura tipo *dueling*, donde se predice una función de ventaja y el valor Q, que mejora *double DQN*	Se utiliza *priorized experience replay*, la arquitectura tiene más parámetros (modelo más complejo)
[Mnih et al., 2016] (*actor-critic*)	Ninguna	RMSProp	para Q-learning	Dos CNNs para V y Q	Diferentes agentes son lanzados en paralelo para realizar exploración utilizando Q-learning de uno y múltiples pasos, SARSA y un métodos tipo *actor-critic*, no se utiliza *memory replay*	Para las propuestas que usan SARSA y Q-learning de uno y múltiples pasos la elección del valor ϵ es una desventaja
[Kaiser et al., 2019]	Ninguna	SGD	No	Autoencoder con diferentes entradas sobre la red	Se propone una arquitectura donde se predice el siguiente frame del juego y la recompensa mediante un autoencoder, converge con pocas instancias	Usa un modelo que copia al emulador para predecir el siguiente frame, utiliza un modelo pre-entrenado para encontrar la copia del emulador de Atari.

Continua en la siguiente página

Tabla 1 – *Continuación*

Algoritmo	Experience Replay	Optimizer	Target Network	Arquitectura de ANN	Principal Aporte	Limitaciones
[Bellemare et al., 2017] (C51)	*Prioritized experience replay*	Adam	Si	CNN con 51 salidas por acción	En vez de utilizar una sola salida por cada acción, se tiene un conjunto de salidas (51) que son una distribución de probabilidad discretizada	Se tiene un modelo con más pesos, por lo que las actualizaciones son más pesadas que otros algoritmos, utiliza *experience replay*
[Dabney et al., 2018b] (IQN)	*Prioritized experience replay*	Adam	Si	CNN	Se presenta un enfoque donde se predicen los valores de salida mediante una distribución de probabilidad utilizando quantiles, se extiende QR-DQN	Se obtiene un modelo más complejo al tener varias salidas por acción, en vez de solo una, se sigue utilizando *experience replay*
[Hessel et al., 2017] (*Rainbow*)	*Prioritized experience replay*	Adam	Si	Dueling Network	Se propone un método que combina las mejoras de DQN utilizando *prioritized experience replay*, una arquitectura tipo dueling, utiliza como salidas distribuciones de probabilidad como C51, entre otras. Es el actual estado del arte	Al combinar este tipo mejoras a DQN puede llegar a ser un método complejo, además de que el uso de *experience replay* sigue siendo importante en estos enfoques

Tabla 2: Trabajos Relacionados a transferencia de conocimiento en DRL

Referencia	Policy Distillation	Nueva tarea	Esquema de transferencia	Modelo Objetivo	Aportes	Limitaciones
[Rusu et al., 2015] (Policy Distillation)	Si	No	$1-1$, $n-n$	Comprimido para $1-1$, más grande para $n-n$	*Policy distillation* es un esquema de transferencia de conocimiento a una red con menos parámetros, así como transferir múltiples tareas a una red	Transfiere a las mismas tareas, no deja que la red alumno aprenda por sí misma como otros enfoques
[Parisotto et al., 2016]	Si	No	$n-1$	Misma arquitectura para base y objetivo	Se propone un método para transferir conocimiento a la misma tarea y un modelo genérico para múltiples tareas	Cuando los modelos (base y objetivo) se entrena una red para transformar a menor dimensión para realizar la transferencia

Continua en la siguiente página

Tabla 2 – *Continuación*

Referencia	Policy Distillation	Nueva tarea	Esquema de transferencia	Modelo Objetivo	Aportes	Limitaciones
[Yin and Pan, 2017]	Si	No	$1-1$	Modelo más complejo que las tareas base	Enfoque basado en *policy distillation*, donde se tiene un conjunto de tareas base, que se dan como entrada una capa intermedia del modelo objetivo y comparten varias capas, al final la capa de salida tiene el conjunto de acciones disponibles	La principal desventaja es que por cada tarea se añade un conjunto de capas, por lo que se obtiene un modelo mucho más complejo a las tareas base.
[Zahavy et al., 2017]	Si	No	$1-1$	Menos unidades que la tarea base	Se presenta un método para realizar transferencia utilizando *experience replay* del modelo experto	No se tiene la certeza de que los parámetros utilizados en los experimentos tengan un buen desempeño.
[Rusu et al., 2016]	No	No	Multitask	Una capa de salida por tarea	Se propone una arquitectura de red neuronal convolucional, donde se añade una capa por cada tarea objetivo, se realiza poda del modelo	Se obtiene un modelo con más pesos por optimizar, hay que entrenar el mismo modelo por cada tarea, no se obtiene el mismo desempeño
[Mittel et al., 2018]	No	Si	Se entrenan dos tareas	Adversarial y se reutiliza parte del modelo	Transferencia de aprendizaje con mapeo de estados en ambas tareas (base y objetivo) utilizando modelo tipo adversarial, se reutiliza parte del modelo y se entrena el agente con un método tipo *actor-critic*	El entrenamiento se realiza en dos pasos, primero se entrena una GAN para encontrar el mapeo de estados y después se entrena el agente *actor critic*, lo cual puede ser más costoso
[Carr et al., 2018]	No	Si	Se entrenan dos tareas	*Adversarial Autoencoder* y se reutiliza parte del modelo	Se utiliza un Autoencoder para encontrar una representación más pequeña de la tarea objetivo y se acelera el proceso de entrenamiento	Se tiene que entrenar el Autoencoder, un modelo generativo que distinga entre la tarea base y la tarea objetivo, y la política de la tarea objetivo.
[de la Cruz et al., 2016]	No	Si	$1-n$	Similar a la tarea base	Se realiza un esquema tradicional *fine-tuning* en capas completamente conectadas mejorando que estos enfoques funcionan para DRL	En los experimentos se realiza transferencia entre dos juegos similares, la forma de selección es intuitiva

Continua en la siguiente página

Tabla 2 – *Continuación*

Referencia	**Policy Distillation**	**Nueva tarea**	**Esquema de transferencia**	**Modelo Objetivo**	**Aportes**	**Limitaciones**
[Fawaz et al., 2018]	No	Si	$1-1$	Similar a la tarea base	Se presenta una métrica basada en similitudes entre conjuntos de datos, la métrica propuesta muestra tener buen desempeño al acertar en la mayoría de las predicciones	Se utiliza para enfoques basados en clasificación y no para DRL, esta basado en similitudes de los conjuntos de entrenamiento y no en el modelo que se utilizará
[Schmitt et al., 2018]	Si	No	$1-1,\ 1-n$	Mismo que la tarea base	Se utiliza una esquema "maestro-alumno", donde el se trata alinear las salidas de ambos modelos, utiliza una función de entropía como función de pérdida	No se hacen experimentos con el baseline de juegos de Atari, por lo que es difícil hacer la comparación
[García et al.]	No	Si	$1-1\ 1-n$	Mismo o con menos unidades	Proponer una medida basada en la arquitectura de CNNs, realizar transferencia de aprendizaje a un modelo reducido basado en términos relevantes de una CNN y aplicar transferencia de aprendizaje con múltiples tareas	Este método no se basa en aplicar transferencia de aprendizaje en la misma tarea como *policy distillation*

4 Propuesta

En esta sección se presenta la propuesta de tesis, primero se presenta la problemática a resolver, junto con los distintos problemas a solventar. Posteriormente, se presentan los objetivos a alcanzar junto con las preguntas de investigación y la hipótesis. Finalmente, se presenta la metodología propuesta para alcanzar los objetivos junto con el plan de trabajo.

4.1 Problemática

Los enfoques basados en aprendizaje profundo tienen la limitación de que se necesitan muchos ejemplos para alcanzar un desempeño aceptable, además de que el proceso de entrenamiento es demasiado aún teniendo equipo especializado (en nuestros experimentos 50 iteraciones tarda un día y medio en entrenar y una iteración son 250,000 instancias de entrenamiento).

El objetivo en este proyecto de tesis es proponer un esquema de transferencia de conocimiento para reducir el tiempo de entrenamiento en agentes DRL, lo cual se traduce en menos ejemplos para entrenar una nueva tarea. Para ello se propone utilizar modelos pre-entrenados para transferir elementos relevantes para la tarea objetivo. A continuación se presentan distintos puntos a tomar en cuenta para alcanzar el objetivo de esta tesis.

Como se menciona en [Taylor and Stone, 2009], un problema importante es seleccionar entre diferentes tareas base para realizar la transferencia de conocimiento en DRL. Para resolver este problema se han propuesto métricas entre MDPs, algunas de ellas son difíciles de implementar para enfoques basados en DRL. En este proyecto se propone una medida de distancia entre tareas, para elegir la tarea base mas adecuada para aplicar la transferencia.

Transferir el modelo completo hacia la tarea objetivo puede resultar contraproducente debido a que se pueden transferir elemento que no sean relevantes para la tarea objetivo. Reutilizar los pesos optimizados para una tarea, que a su vez sean relevantes para la tarea objetivo, puede ayudar a reducir el tiempo del entrenamiento de CNNs. Hacer una selección de los parámetros a transferir puede ser mejor que transferir todos los parámetros, ya que no todos pueden aportar información. Transferir los elementos relevantes de múltiples tareas para una nueva tarea puede reducir aún más el tiempo de entrenamiento de dicha tarea para alcanzar mejor recompensa.

Los enfoques propuestos al aplicar TL en DRL se basan en entrenar un modelo más pequeño para utilizarlos en dispositivos con pocos recursos computacionales utilizando la misma tarea como base y objetivo, además existen enfoques donde se crea un modelo para diferentes tareas. Sin embargo, en esta tesis se plantea transferir estos conocimiento para la una tarea objetivo diferente a la base.

Los problemas mencionados anteriormente serán los puntos a resolver durante este proyecto, el cual tiene como fin realizar transferencia de aprendizaje hacia una nueva tarea en DRL. Usar técnicas basados en TL para DRL es un enfoque que no ha sido muy utilizado, ya que es un tema reciente. Debido a esto, los enfoques que se han propuesto no se centran en transferir hacia una nueva tarea y los pocos que existen la selección entre tareas se realizan de manera subjetiva.

4.2 Preguntas de Investigación

Teniendo en cuenta que se tiene un conjunto de tareas base $\mathcal{T} = t_1, t_2, \ldots, t_n$ y una tarea objetivo t_o a la que se le aplicará la transferencia de aprendizaje. Las preguntas de investigación son las siguientes:

- ¿Cómo seleccionar una tarea base del conjunto $\mathcal{T}$ o un subconjunto de estas, que tendrá mejor desempeño al hacer transferencia de conocimiento hacia la tarea objetivo?
- ¿La selección de elementos relevantes en las tareas seleccionadas será mejor que transferir todos los elementos del modelo asociado a dicha tarea para reducir el riesgo de transferencia negativa y ayudar a mejorar la recompensa obtenida con menos instancias?
- Tomando en cuenta que las tareas (q, v, ambas o π) en S se aproximaron mediante redes neuronales profundas, ¿Cómo se puede transferir elementos de múltiples CNNs hacia la tarea objetivo sin importar la arquitectura entre ellas?
- ¿Se puede determinar si un modelo donde los pesos iniciados de manera aleatoria tendrá mejor desempeño que al realizar la transferencia de tareas base?

4.3 Hipótesis

La transferencia de elementos relevantes de redes neuronales convolucionales profundas utilizando distintas tareas base reduce el tiempo de convergencia a recompensas obtenidas en entrenamiento sin transferencia de conocimiento y puede mejorar la recompensa obtenida al final del entrenamiento.

4.4 Objetivos

En esta sección se presentan el objetivo general y los objetivos específicos de este proyecto.

4.4.1 Objetivo General

El objetivo general de este proyecto es proponer, diseñar, desarrollar y validar un esquema de transferencia de conocimiento en aprendizaje por refuerzo profundo, basado en la arquitectura de redes neuronales convolucionales profundas que acelere la convergencia en la recompensa obtenida por el agente en un 75% .

4.4.2 Objetivos Específicos

Los objetivos específicos son:

1. Proponer una medida entre tareas basada en la arquitectura de redes neuronales convolucionales profundas, para determinar cuál tendrá mejor desempeño al aplicar el entrenamiento utilizando transferencia de aprendizaje en la tarea objetiva, sin riesgo de transferencia negativa.
2. Validar experimentalmente que la medida propuesta realice la predicción de cuál modelo base tendrá desempeño equivalente reduciendo el tiempo de entrenamiento comparado con una inicialización de pesos aleatorios.
3. A partir de la medida propuesta y estadísticas, determinar los elementos relevantes de la CNN asociada a la tarea base, para transferirlos a la tarea objetivo.
4. Determinar en que casos utilizar un modelo con pesos aleatorios será mejor que transferir de alguna tarea base.

5. Proponer un esquema de transferencia utilizando múltiples tareas base basado en los elementos relevantes de las tareas base utilizando la medida propuesta y otras estadísticas.

6. Evaluar si el enfoque propuesto también es aplicable en problemas de clasificación basada en aprendizaje profundo.

7. Evaluar el desempeño de la transferencia tomando en cuenta el número de episodios para alcanzar un desempeño aceptable.

4.5 Metodología

Basado en los objetivos planteados anteriormente, se propone la siguiente metodología, la cual es detallada en los siguientes puntos y se puede observar gráficamente en la Figura 14. Los objetivos se basan en proponer métodos de transferencia de conocimiento con diferentes esquemas $(1-1, n-1)$ en algoritmo de DRL.

1. Realizar una revisión sobre dos temas importantes para este proyecto, los cuales son transferencia de conocimiento en algoritmos basados en DRL, esto debido a que son los enfoques con los que se comparará nuestra propuesta, además de los enfoques propuestos sobre transferencia en algoritmos de clasificación basados en DL. Esto servirá para conocer en que consisten los enfoques que se han propuesto sobre estos temas, así como sus beneficios y limitaciones.

2. Proponer una medida entre tareas. Esta medida servirá para determinar dentro de un conjunto de tareas base cuál o cuáles tareas servirán para realizar transferencia de conocimiento hacia la tarea objetivo. Para esta medida se tomaran en cuenta los siguientes puntos.

   (a) Salida de una capa oculta. A diferencia de [Fawaz et al., 2018] donde se comparan las instancias directamente utilizando una reducción de dimensiones, nuestro enfoque utilizar la representación obtenida por el modelo pre-entrenado.

   (b) Comparación entre espacios de acciones. El espacio de acciones de los MDPs es un factor importante, encontramos empíricamente que entre más parecidos sean dichos espacios se tendrá mayor éxito al realizar la transferencia de conocimiento.

   (c) Mezcla de los dos puntos anteriores. Tomando en cuenta que los dos puntos anteriores son importantes para las tareas, utilizar las dos partes puede servir para que la medida propuesta sea más exacta al seleccionar la tarea base.

3. Transferencia selectiva de elementos relevantes. Existen diferentes métodos de compresión de redes neuronales como se muestra en [Cheng et al., 2017]. La principal limitación que encontramos es que las tareas base y objetivo son la misma. Para este proyecto se propondrá un enfoque que seleccione elementos relevantes (pesos, partes de red, *kernels*, pesos) del modelo asociado a la tarea base para una nueva tarea objetivo.

4. Transferencia de múltiples tareas hacia una tarea objetivo. Utilizar los dos términos anteriormente propuestas para transferir elementos relevantes de diferentes tareas hacia una nueva tarea. En este caso, se propone que utilizar elementos de diferentes tareas base puede ayudar a reducir el tiempo de entrenamiento que si solo se transfirieran los elementos de una sola tarea base.

5. Aplicar el enfoque propuesto en otros dominios. Muchos de los enfoques propuestos utilizando técnicas de TL son desarrollados para clasificación (DL) o para RL, nosotros planteamos realizar pruebas para ambos enfoques (clasificación o RL) para observar que tan robusto es nuestro enfoque.

4.6 Plan de Trabajo

El cronograma de actividades se presenta en la Tabla 3 utilizando periodos de cuatro meses en cada año. A continuación se presentan de manera detallada sobre cada actividad para alcanzar los objetivos propuestos.

1. Realizar revisión de literatura sobre los temas referentes a este trabajo. El primer paso de esta metodología es realizar una revisión de literatura sobre varios temas referentes a este trabajo. Los temas más relevantes son:

    - Algoritmos DRL. Estos algoritmos son importantes porque se utilizaran como base para realizar transferencia de aprendizaje, algunos de los algoritmos que se pretenden utilizar son DQN, C51 y Rainbow, este último el actual estado del arte.
    - TL en DRL. Este es un tema de interés que esta comenzando a abordarse, por lo que es una gran área de oportunidad. Este tema es importante debido a que algunos de estos serán utilizados para realizar comparación de resultados contra el enfoque propuesto.
    - TL en DL. Aunque es diferente a los enfoques basados en DRL, en este tema existen más trabajos reportados en la literatura. ESto puede ayudar a determinar si los modelos encontrados por estos enfoques propuestos son robustos a tareas sobre DRL.

2. Proponer una medida de distancia entre tareas para predecir cual tendrá mejor desempeño al realizar transferencia de conocimiento a la tarea objetivo. El objetivo de este trabajo es proponer un método para realizar transferencia de conocimiento de múltiples tareas a una nueva tarea. Por lo que el primer paso es predecir cuál tarea base tendrá mayor éxito al entrenar un nuevo modelo en la tarea objetivo basándose en elementos relevantes de la tarea como: arquitectura de la red neuronal, espacio de acciones, función de recompensa, función de transición, entre otras.

3. Transferencia basada en *fine-tuning*. Fine-tuning es una técnica de transferencia de conocimiento donde se reutilizan los pesos de un modelo pre-entrenado y se utilizan como punto de partido para entrenar un modelo en una nueva tarea. Se propone realizar dos enfoques basados en esta técnica, los cuales se presentan a continuación.

    - Realizar transferencia con el esquema $1 - 1$ donde el modelo de una tarea base es utilizado para transferir los pesos del modelo a una tarea objetivo. Este enfoque se propone para validar que la medida propuesta funcione de manera correcta, este tipo de transferencia ya ha demostrado tener buen desempeño en distintas tareas de clasificación y en algunos experimentos sobre DRL.
    - Realizar transferencia selectiva de pesos. En algunos caso puede resultar contraproducente transferir el modelo completo, por lo que se pretende realizar transferencia de elementos relevantes de la red neuronal entrenada en la tarea base, para ello se pretenden usar algunas métricas estadísticas para determinar cuáles elementos transferir. Al realizar esta selección se pueden realizaran dos enfoques:

  - Tener como resultado un modelo más pequeño en la tarea base. En este caso transferir una parte de los pesos de la red neuronal base teniendo como resultado un modelo más pequeño que necesita menos instancias para aprender.
  - Reemplazar los elementos que no aportan mucha información con pesos aleatorios. Para realizar esta transferencia se inicia el entrenamiento con algunos pesos aleatorios y se tiene un modelo con del mismo tamaño que la tarea base.

4. Transferencia de múltiples tareas. Tomando en cuenta el punto anterior, transferir elementos relevantes de distintas tareas puede mejorar la transferencia respecto a la transferencia de una sola tarea base. Hay que tener en cuenta dos enfoques que se pueden abordar, los cuales se muestran a continuación.

   - Utilizar la misma arquitectura para todas las tareas base. Este enfoque se aborda en la mayoría de los trabajos reportados. Debe tomarse en cuenta que las arquitecturas tienen el mismo número de capas y elementos en sus redes neuronales. La arquitectura de la tarea objetivo puede o no ser la misma que las tareas base.
   - Utilizar redes neuronales asociadas a las tareas base con arquitecturas diferentes, en este enfoque se deben resolver varios problemas, algunos son los siguientes: cuál será la arquitectura de la red neuronal de la tarea objetivo, cuántos elementos transferir en cada capa, qué elementos va a aportar cada tarea base, entre otros. Para resolver este problema se utilizaran los enfoques utilizados en los puntos anteriores.

5. Pruebas en otros dominios. En este punto se pretende utilizar un dominio diferente al utilizado en los puntos anteriores, en principio se utilizaran juegos de Atari para realizar transferencia de conocimiento. Algunos dominios que se pueden utilizar son los siguientes: para enfoques basados en DRL, se pueden utilizar los clásicos problemas de péndulos o aplicaciones en robótica, mientras que para DL, se pueden utilizar enfoques de visión por computadora (que puede estar relacionado a aplicaciones en robótica), o problemas de clasificación.

6. Validación. La evaluación de los enfoques propuestos se realizan mediante algunas métricas propuestas en [Machado et al., 2018], las cuales son.

   - *Jumpstart*. Es la ganancia de desempeño inicial al realizar transferencia de conocimiento.
   - Mejor política. La recompensa de la mejor política obtenida durante el entrenamiento.
   - Desempeño final. Recompensa obtenida al final del entrenamiento.

Tabla 3: Cronograma de actividades

	Año															
Actividad	**2018**				**2019**				**2020**				**2021**			
Periodo	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Revisión de literatura	X	X	X	X	X	X	X	X	X	X	X	X				
Redacción de propuesta			X	X												
Métrica basada en acciones	X	X														
Métrica basada en modelo		X	X													
Métrica combinada			X	X												
Transferencia con *fine-tuning*				X	X	X										
Transferencia selectiva de pesos						X	X	X								
Transferencia de múltiples tareas							X	X	X							
Transferencia a un modelo reducido								X	X	X						
Transferencia con arquitecturas diferentes									X	X	X					
Pruebas en otros dominios										X	X	X				
Experimentos y evaluación		X	X			X	X		X	X		X	X			
Escritura de artículos							X				X				X	
Escritura de tesis			X	X	X	X	X	X	X	X	X	X	X	X		
Defensa de tesis														X		

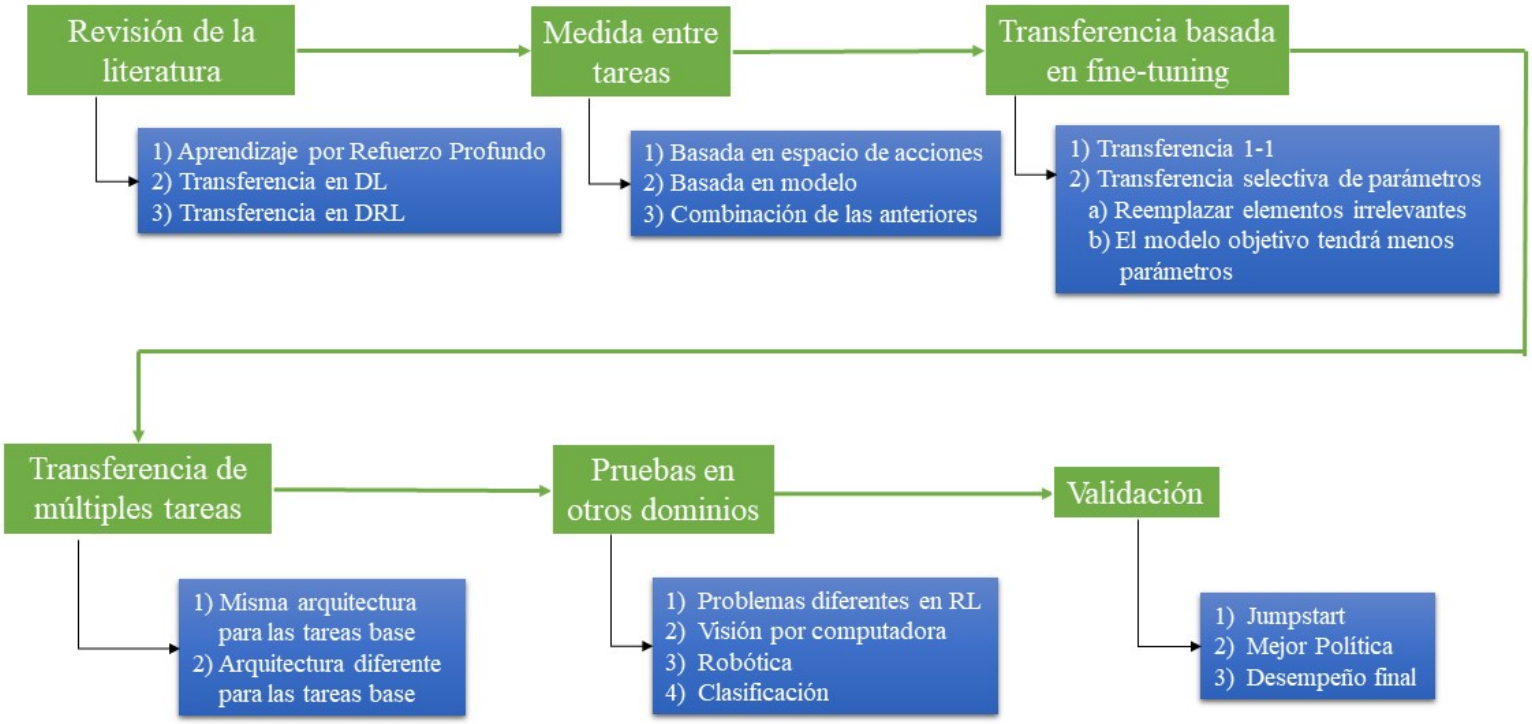


Figura 14: Metodología propuesta en este trabajo.

4.7 Contribuciones Esperadas

Las contribuciones esperadas en este trabajo son las siguientes:

- Una medida entre tareas basada en elementos relevantes de dichas tareas.
- Un esquema de transferencia de conocimiento basado en *fine-tuning*.
- Un esquema de selección de parámetros relevantes de una red neuronal, para realizar transferencia de conocimiento.
- Un esquema de transferencia de múltiples tareas a una nueva aplicando los puntos anteriores.

5 Resultados Preliminares

En esta sección se muestran los resultados preliminares sobre una medida de distancia entre tareas, la cual se basa en la salida de una capa oculta de una red neuronal, además de la comparación entre conjuntos de acciones, finalmente se propone una combinación de estas. La medida propuesta se valida mediante la transferencia basada en *fine-tuning* utilizando el dominio de juegos de Atari con el algoritmo DQN.

5.1 Consideraciones del Entrenamiento

En esta sección se presenta la medida entre tareas para seleccionar en el conjunto de $\mathcal{T}$. Primero, se muestran notación y definiciones que se utilizan en esta sección. Una tarea es considerada como la función de clasificación (regresión, que en este caso es CNN) asociado a un juego de Atari.

Muchas consideraciones que sobre el ambiente de juegos de Atari se reportan en el artículo [Machado et al., 2018], en los siguientes párrafos se mencionan algunas de las estas consideraciones.

Existen dos formas de establecer cuando termina un episodio: una es que el episodio finaliza una vez que se pierden todas las vidas, mientras que la otra es cuando se pierde una vida. En este trabajo un episodio se termina cuando el agente pierde todas las vidas.

Las acciones asociadas a cada tarea son las combinaciones disponibles de las direcciones del *joystick* (palanca) y un botón, las acciones están en un rango entre tres y dieciocho dependiendo el juego. Las acciones disponibles para cada juego se pueden observar en la Tabla 20.

Para los estados se realiza el mismo pre-procesamiento del artículo original DQN [Mnih et al., 2015] donde se aplica una transformación a escala de grises y se da como entrada cubos de cuatro imágenes, este cubo de imágenes es considerado como un estado.

La recompensa puede tener tres valores: cuando no hay cambios en la puntuación se establece un valor de 0, mientras que cuando hay un cambio positivo se da un valor de 1 y cuando hay un cambio negativo se establece un valor -1. Esto se realiza para que no haya cambios bruscos en los pesos de la CNN y no haya tanta diferencia entre las puntuaciones de cada juego.

Para reportar los resultados Machado propone realizar el entrenamiento de la CNN en un número establecido de pasos para posteriormente evaluar el modelo con determinado número de pasos. Este esquema se sigue para reportar los resultados y se presentan en forma de curvas de aprendizaje.

5.2 Medida de espacio de acciones y la salida de una capa oculta

La medida propuesta se basa en realizar una comparativa de la salida de una CNN entrenada en la tarea base y que tiene un buen desempeño en dicha tarea. Además, de tomar en cuenta qué tan distintos son los espacios de acciones. En esta sección se presenta la medida de distancia entre tareas.

Para encontrar la medida se utiliza un conjunto de tareas base $\mathcal{T} = \{t_1, t_2, t_3, \ldots, t_n\}$, donde cada tarea $(t_1, t_2, ..., t_n)$ es una DQN entrenada en un juego que estima el valor Q para cada acción de dicha tarea y tiene un buen desempeño basado en la recompensa obtenida. El objetivo es encontrar la tarea objetivo t_o es un agente DRL, utilizando el conocimiento obtenido en la DQN de las tareas base.

Al tener un conjunto de tareas base, elegir entre todas ellas es un problema importante para determinar cuál de ellas servirá para el entrenamiento de la tarea objetivo. Para resolver este problema, en este trabajo se propone una medida de distancia entre tareas basada en la salida de una capa oculta que encuentra similitudes visuales entre el conjunto $\mathcal{T}$ y la tarea t_o y una comparación entre espacios de acciones (Ecuación 17).

Los principales beneficios de la medida propuesta es que se puede predecir el modelo obtenido que tendrá mejor desempeño respecto al entrenamiento iniciando con pesos aleatorios. Al reducir el tiempo de convergencia a recompensa máxima obtenida se necesitan menor número de instancias para alcanzarla.

La métrica utiliza el universo de acciones, que es la unión entre las acciones del conjunto de tareas base $\mathcal{T}$ y la tarea objetivo t_o, como se muestra en la Ecuación 16.

$$\mathcal{U} = \{A_1 \cup A_2 \cup A_3 \cup \cdots \cup A_n \cup A_o\} \tag{16}$$

$$Sim(t_k, t_o) = \alpha \cdot dist_1(r_k, r_o) + (1 - \alpha) \cdot dist_2(A_k, A_o) \tag{17}$$

La primera parte de la Ecuación 17 está asociada a las similitudes visuales entre las tareas del conjunto $\mathcal{T}$ y t_o. En esta parte de la medida utiliza la salida de alguna capa oculta de la CNN del conjunto $\mathcal{T}$. Para llegar a ella se realizan los siguientes pasos (y se puede ver gráficamente en la Figura 15):

1. Se tienen de entrada k modelos entrenados asociados a diferentes tareas del conjunto $\mathcal{T}$.
2. De estos modelos se extraen las salidas de una capa oculta, por lo que se tiene como salida un vector de atributos numéricos, para cada estado.
3. Se genera una serie de episodios con una política $\epsilon - greedy$ utilizando el modelo pre-entrenado (o bien pueden ser partidas generados por humanos).
4. Para la tarea base también se genera una serie de episodios con una política $\epsilon - greedy$ utilizando la DQN entrenada.
5. Las imágenes se dan como entrada a la DQN asociada a la tarea base para extraer vectores numéricos de dichas instancias (de ambas tareas base y objetivo), estas características son utilizadas para obtener la distancia entre las tareas.
6. Se obtiene una muestra de n elementos para cada tarea.

7. Se encuentra una matriz de distancias euclidianas (ecuación 18) de manera exhaustiva para entre las instancias de la tarea base y la tarea objetivo.

8. Se obtiene el promedio de los elementos en la matriz de distancias del paso anterior.

9. Se realiza p veces el paso anterior con una muestra aleatoria y se encuentra el promedio de las distancias entre la tarea base y objetivo.

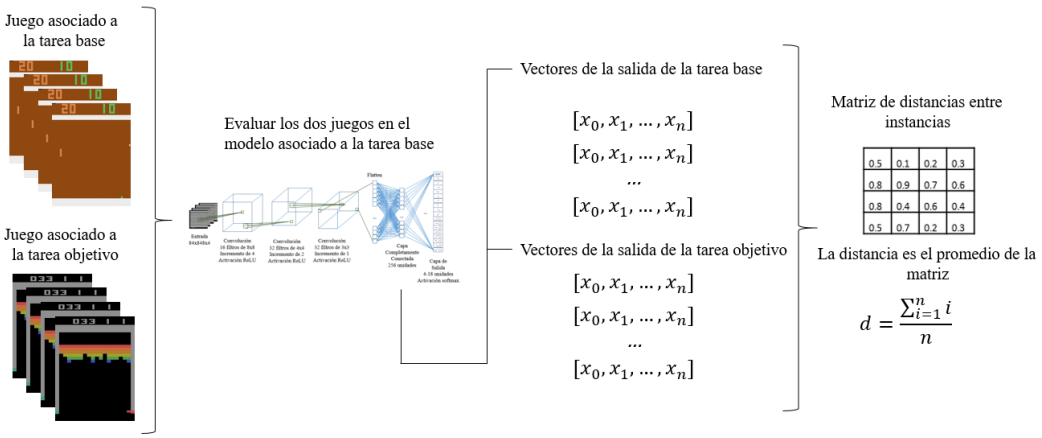


Figura 15: Diagrama que representa los pasos para encontrar la primera parte de la Ecuación 17.

$$d_1(r_k, r_o) = \sqrt{(r_{k1} - r_{o1})^2 + (r_{k2} - r_{o2})^2 + (r_{k3} - r_{o3})^2 + ... + (r_{kn} - r_{on})^2} \tag{18}$$

La segunda parte de la Ecuación 17 indica cuántas acciones compatibles existen entre la tarea base y la tarea objetivo. Si los dos conjuntos son disjuntos, la d_2 será el valor máximo que para este caso es 2. De lo contrario, el primer criterio es que entre mayor sea la intersección de las acciones, la distancia tiene que ser menor.

$$d_2(A_k, A_o) = \begin{cases} 2 & si \quad A_k \ y \ A_o \ son \ disjuntos \\ \dfrac{1}{|A_k \cap A_o|} + \dfrac{|A_o - A_k| + 1}{A_{\mathcal{U}} - |A_k - A_o|} & en \ otro \ caso \end{cases} \tag{19}$$

Con la Ecuación 19 se compara que tan parecido es el espacio de acciones entre tareas. Lo que se busca con esta ecuación es que las tareas más parecidas sean aquellas que tengan la mayor parte de las acciones para t_o, además de tomar en cuenta las acciones que difieren entre conjuntos, así como el universo de acciones (factores que no toman en cuenta otras medidas como en indice Jaccard).

El comportamiento de la segunda parte de la Ecuación 19 se puede observar en la Figura 16 desde diferentes ángulos. Los ejes x y y representan los valores de las cardinalidades de las diferencias entre los conjuntos de acciones ($A_o - A_k$ y $A_k - A_o$, respectivamente), mientras que el eje z muestra es el valor de la distancia utilizando $|\mathcal{U}|$=100.

En dicha Figura se puede observar que cuando los espacios de acciones son similares la distancia entre ellas es menor, mientras que cuando alguna de las dos diferencias es muy grande y la otra muy pequeña el resultado de la Ecuación 19 es mayor.

Las dos partes de la ecuación se debe buscar el valor del parámetro α (de la ecuación 17) que se ajuste mejor para encontrar el modelo de las tareas base y que tendrá mejor desempeño al realizar la transferencia.

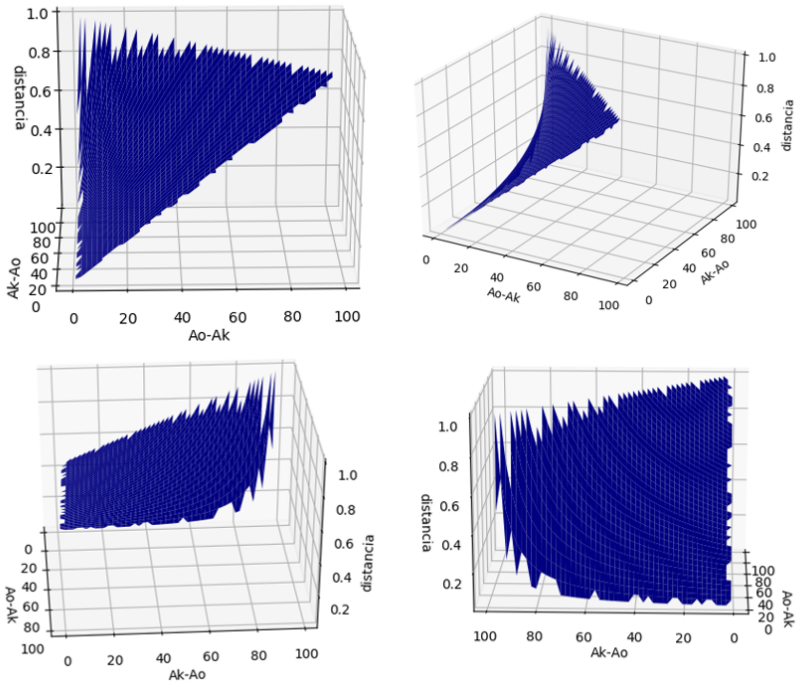


Figura 16: Comportamiento de la segunda parte de la ecuación 19 con diferentes vistas.

La medida propuesto captura dos cosas importantes de las tareas. Primero en venz de comparar los estados de que se dan como entrada a la DQN (como en [Fawaz et al., 2018]) se comparan las salidas del modelo a transferir. La segunda es la comparación de las salidas de la DQN las cuales son los espacios de acciones entre tareas.

5.3 Resultados experimentales

En estos experimentos se utiliza el dominio de juegos de Atari. EL algoritmo de DRL que se utiliza es DQN, debido a que fue el primer algoritmo que tuvo buen desempeño y a que existen diversas implementaciones del mismo. La arquitectura utilizada es la misma presentada en el artículo [Mnih et al., 2015] (ver figura 17).

En estos experimentos se utilizaron modelos pre-entrenados que han sido proporcionados por los autores de la biblioteca Dopamine-rl [Castro et al., 2018]. Las curvas de aprendizaje con el algoritmo DQN con cinco iteraciones se muestran en la Figura 21.

Se tienen modelos entrenados para todos los juegos mostrados en la Tabla 20, junto con sus acciones disponibles. Todos los juegos fueron entrenados con 200 iteraciones. Cada iteración se entrenaron con 250,000 instancias, seguida de 125,000 ejemplos para la evaluación con ϵ=0.001.

Para la primera parte de la Ecuación 17 se utiliza la penúltima capa de la CNN, donde se tiene una salida de 512 valores numéricos al utilizar la misma arquitectura de la Figura 17. Como se puede observar en la Ecuación 17 se tiene que ajustar el valor del α. Se utilizaron

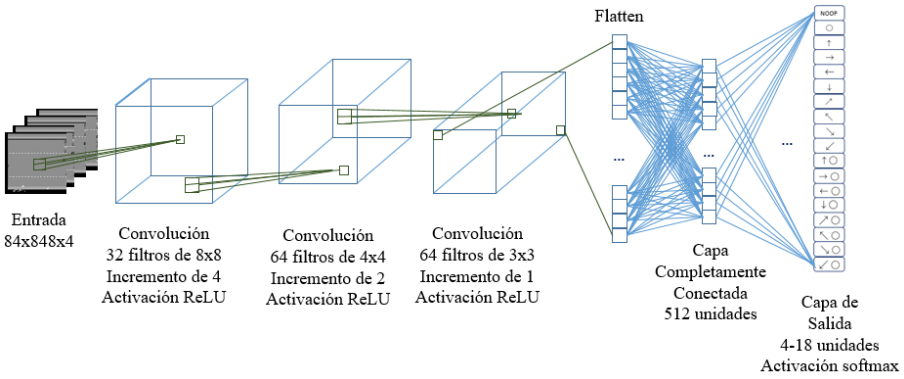


Figura 17: Arquitectura de la CNN utilizada en [Mnih et al., 2015].

valores estáticos para el valor α y otros dinámicos utilizando el promedio de las Ecuaciones 18 y 19 como en la Ecuación 20. Los ajustes propuestos son los siguientes:

- Se da todo el peso a la salida de la CNN $\alpha = 1$ (Figura 18a).
- Se da todo el peso a la comparación de las acciones $\alpha = 0$ (Figura 18b).
- Se da el mismo peso a las dos partes de la medida $\alpha = 0.5$ (Figura 18c).
- Se da más peso a la salida de la CNN $\alpha = 0.75$ (Figura 18d).
- Se da más peso a la comparación entre conjunto de acciones $\alpha = 0.25$ (Figura 18e).
- Se ajusta con el promedio de los resultados de la comparación entre conjuntos de acciones (Figura 18f).
- Se ajusta con el promedio de las salidas de la capa oculta (Figura 18g).

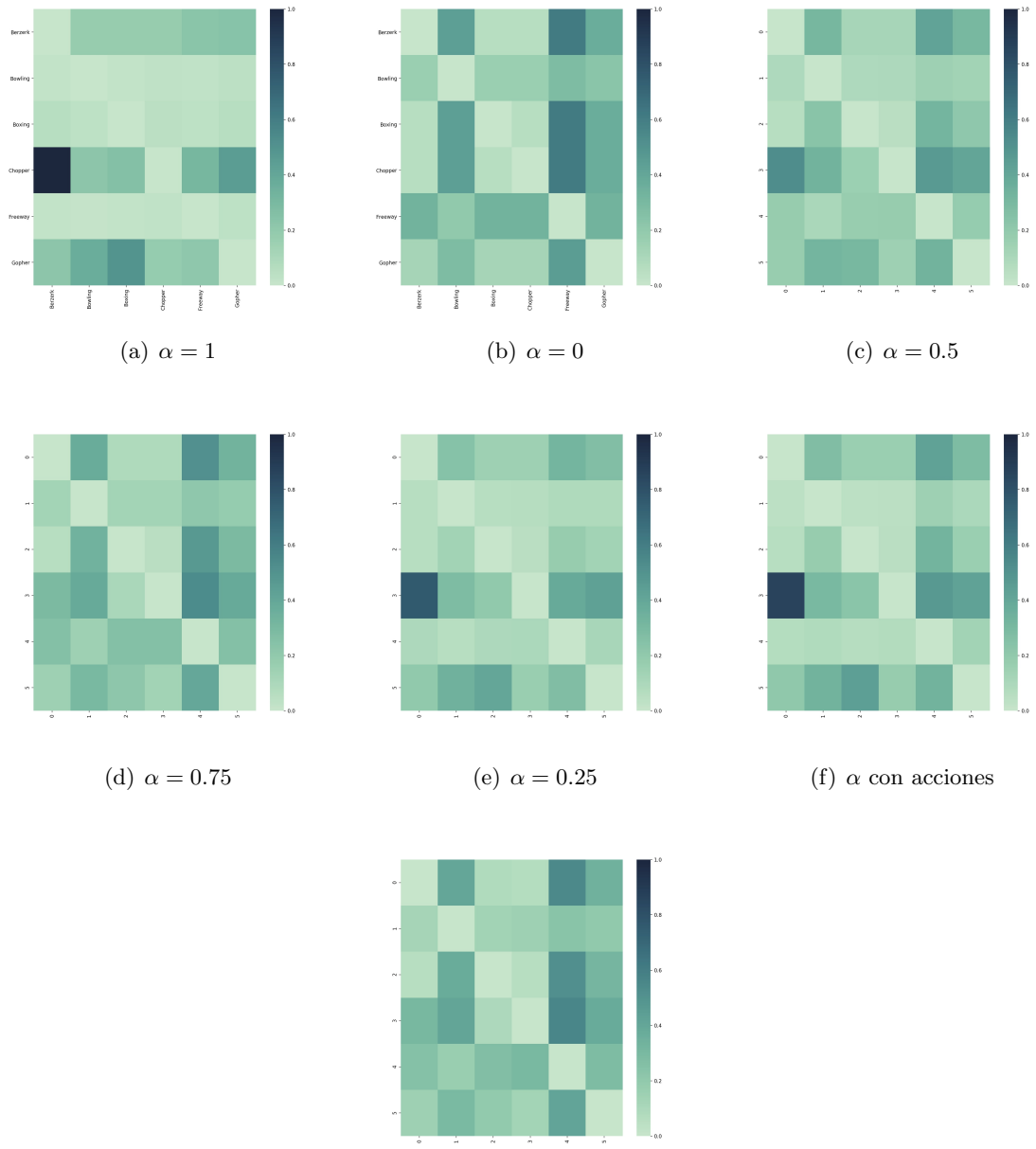
$$alpha = \frac{1}{n}\sum_{i=1}^{n} d(A_t, A_{si}) \tag{20}$$

Al aplicar la transferencia basada en *fine-tuning* se transfirió el modelo completo, excepto la última capa, donde se inician los pesos con valores aleatorios pequeños. El proceso de entrenamiento es el mismo que si los pesos se inicializaran con números aleatorios, se usan los mismos parámetros que en [Mnih et al., 2015].

Se usó la biblioteca Dopamine-rl [Castro et al., 2018] y se corrieron cincuenta iteraciones para cada experimento, una iteración corresponde al entrenamiento con 250,000 instancias, y cada punto en la gráfica corresponde al promedio de puntuación obtenida después de evaluar con 125,000 instancias.

Se eligieron seis juegos para realizar la transferencia de aprendizaje: dos donde se aprende rápido (*Freeway, Boxing*), dos donde es difícil aprender (*Berzerk, Bowling*) y dos donde el proceso de aprendizaje sea promedio (*Chopper Command, Gopher*).

Las curvas de aprendizaje de los experimentos se muestran en la Figura 19, donde se muestra el promedio de dos corridas con cada posible combinación de transferencia de aprendizaje (los resultados completos se muestran en el Apéndice B). En color negro se observa el promedio de las dos mejores curvas de aprendizaje conforme a los experimentos proporcionados en



(a) $\alpha = 1$ (b) $\alpha = 0$ (c) $\alpha = 0.5$

(d) $\alpha = 0.75$ (e) $\alpha = 0.25$ (f) α con acciones

(g) α con Salida

Figura 18: Heatmaps de los resultados al buscar el ajuste del valor α en la Ecuación 17.

la biblioteca Dopamine-rl [Castro et al., 2018] (en el Apéndice B se muestran qué curvas se eligieron).

Debido a que las curvas de aprendizaje pueden ser difíciles de leer o interpretar se presentan distintas métricas de evaluación propuestas en [Machado et al., 2018]. Primero se presenta el *jumpstart* que dice cuanta ganancia en la recompensa hay al inicio del entrenamiento (Tabla 4). El siguiente criterio es el desempeño en la última iteración respecto al *baseline*, estos resultados se pueden observar en la Tabla 5. Finalmente, en la Tabla 6 se puede observar la mejor política obtenida durante las etapas de evaluación. Para *jumpstart* los valores superiores a cero son mejora en el entrenamiento, mientras que en las otras dos métricas los números mayores a uno indican una mejora respecto al *baseline*. Adicionalmente, los superínidices (1) (2) corresponden a los mejores entrenamiento al realizar la transferencia de conocimiento.

Acorde a los resultados se puede observar que en la mayoría de los experimentos se tiene una transferencia positiva. La mejor configuración de la métrica fue cuando se estableció el valor α utilizando los valores dinámicos con la comparación de espacios de acciones.

Acorde en las métricas mostradas se puede concluir: las tareas seleccionadas en mejora respecto al entrenamiento inicial como lo muestra *fine-tuning* en tres de los juegos se seleccionó el mejor desempeño inicial. Basado en la mejor política durante el entrenamiento se seleccionaron modelos donde se encuentra una política mejor respecto a iniciar con pesos aleatorios dentro de las dos mejores tareas seleccionadas. Finalmente, respecto a la política final, en la mayoría de los casos se obtiene una peor política (excepto en Gopher), por lo que no importa de que tarea transferir.

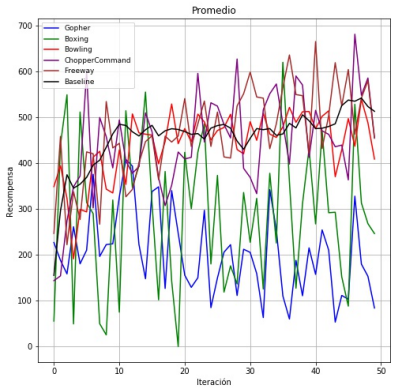
Las curvas de aprendizaje mostradas en la Figura 19 muestran un comportamiento diferente respecto a las curvas *baseline*. Las curvas de transferencia muestran un comportamiento donde se tiene muchos picos en la evaluación del modelo. La situación anterior se puede deber a dos factores: el primero es que cambios pequeños en la CNN pueden cambiar mucho la recompensa obtenida como se menciona en [Mnih et al., 2015] y el otro es que los pesos de la red están optimizados para una tarea diferente, lo que puede atenuar aún más la situación antes mencionada.

Tabla 4: Resultados de la métrica *jumpstart*.

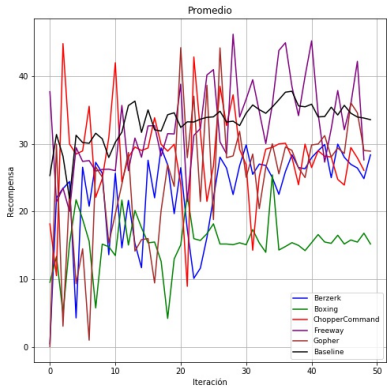
	Berzerk	Bowling	Boxing	Chopper	Freeway	Gopher
Berzerk	X	-0.6587	-0.3168	-0.2349	0.3400	-0.0417
Bowling	0.3572	X	0.0282	-0.2891	0.0318	0.0425
Boxing	-0.1840	-0.4176	X	-0.1991	0.0016	-0.1026
Chopper	-0.0214	-0.1893	-0.0294	X	-0.0011	-0.0148
Freeway	0.1688	0.3280	0.0279	-0.0307	X	0.0535
Gopher	0.1324	-0.6689	-0.3354	-0.0821	0.1115	X

Tabla 5: Resultados de la métrica tomando en cuenta el desempeño de la mejor política.

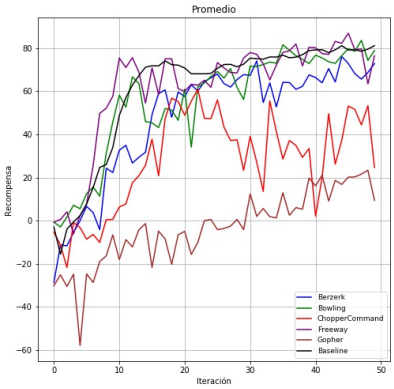
	Berzerk	Bowling	Boxing	Chopper	Freeway	Gopher
Berzerk	X	0.7931	0.9375	1.1720	1.0140	0.8705
Bowling	0.7969	X	1.0305	1.2571	1.0008	1.6984
Boxing	1.1439	0.6728	X	0.7867	0.4964	0.0007
Chopper	1.2568	1.1865	0.7508	X	1.0145	0.6662
Freeway	1.2273	1.2232	1.0713	1.2142	X	1.2882
Gopher	0.7529	1.1706	0.2879	0.7255	0.4976	X



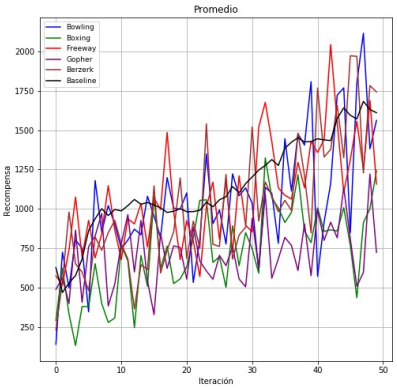
(a) Berzerk



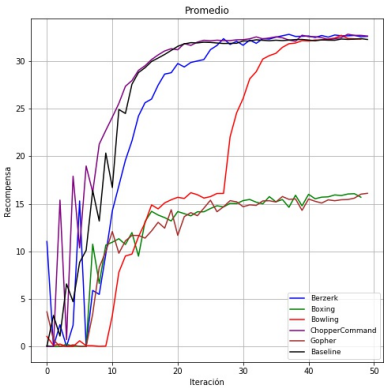
(b) Bowling



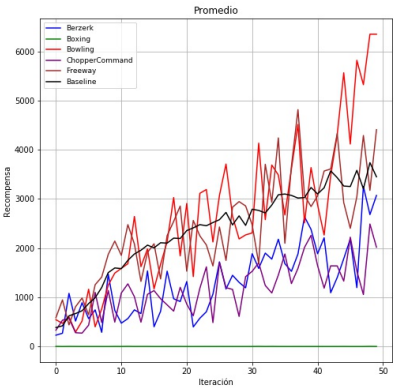
(c) Boxing



(d) Chopper Command



(e) Freeway



(f) Gopher

Figura 19: Curvas de aprendizaje con 50 iteraciones.

Tabla 6: Resultados de la métrica tomando en cuenta el desempeño final.

	Berzerk	Bowling	Boxing	Chopper	Freeway	Gopher
Berzerk	X	0.8452	0.8978	1.0820	1.0100	0.8902
Bowling	0.7969	X	0.9722	0.9696	1.0008	1.8402
Boxing	0.4802	0.4526	X	0.7706	0.4862	0.0002
Chopper	0.8882	0.7770	0.3036	X	1.0110	0.5850
Freeway	0.8856	0.8228	0.9428	0.7186	X	1.2778
Gopher	0.1636	0.8615	0.1153	0.4490	0.4989	X

6 Conclusiones y Trabajo Futuro

En este trabajo se presentan los avances sobre el enfoque propuesto para la transferencia de conocimiento basado en arquitectura, donde se utiliza *fine-tuning* reutilizando los pesos optimizados para una tarea base hacia una tarea objetivo.

Se propone una medida para determinar cuáles tareas del conjunto $\mathcal{T}$ funcionaran para acelerar el proceso de entrenamiento de la tarea objetivo. Esta medida se basa en similitudes visuales capturadas por una DQN pre-entrenada y las diferencias entre los espacios de acciones.

En la mayoría de los experimentos presentados se obtuvo transferencia positiva basado en la mejor política obtenida durante el entrenamiento. Si se toma en cuanta la política final el desempeño no es el mejor a diferencia del desempeño inicial de la red neuronal donde se predice la mejor tarea en la mayoría de los casos.

Las curvas de aprendizaje mostradas muestran un comportamiento caótico utilizando la transferencia de el modelo completo hasta la última capa oculta, esto se puede deber a diferentes factores como que los pesos son optimizados para una tarea diferente, que las características extraídas son diferentes a las que se necesitaría para el juego objetivo o que se están transfiriendo elementos irrelevantes para la tarea objetivo.

Como trabajo futuro se realizaran pruebas con otros algoritmos de DRL como Rainbow, C51 o IQN que han mostrado aprender más rápido que DQN. Utilizar otros parámetros como el algoritmo de optimización de pesos o la forma de seleccionar las instancias que pueda mejorar el comportamiento de las curvas de aprendizaje. Además, proponer una selección de los elementos a transferir utilizando distinta métricas y estadísticas.

Además de utilizar los enfoques propuestos para realizar transferencia de conocimiento utilizando múltiples tareas hacia una tarea objetivo. Algunas ideas que se tienen son encontrar elementos relevantes de las redes neuronales asociadas a las tareas base que para crear una nueva red y transferirlos hacia la nueva tarea.

Agradecimentos

Los autores agradecen al Laboratorio Nacional de Supercómputo del Sureste de México(LNS), perteneciente al padrón de laboratorios nacionales CONACYT, por los recursos computacionales, el apoyo y la asistencia técnica brindados, a través del proyecto No. 201901047c.

A Juegos de Atari

OpenAIgym es la biblioteca que es utilizada por el paquete Dopamine-RL [Castro et al., 2018] la cual se usa para los experimentos mostrados en esta tesis, el espacio de acciones para todos los juegos disponibles se puede observar en la Tabla 20.

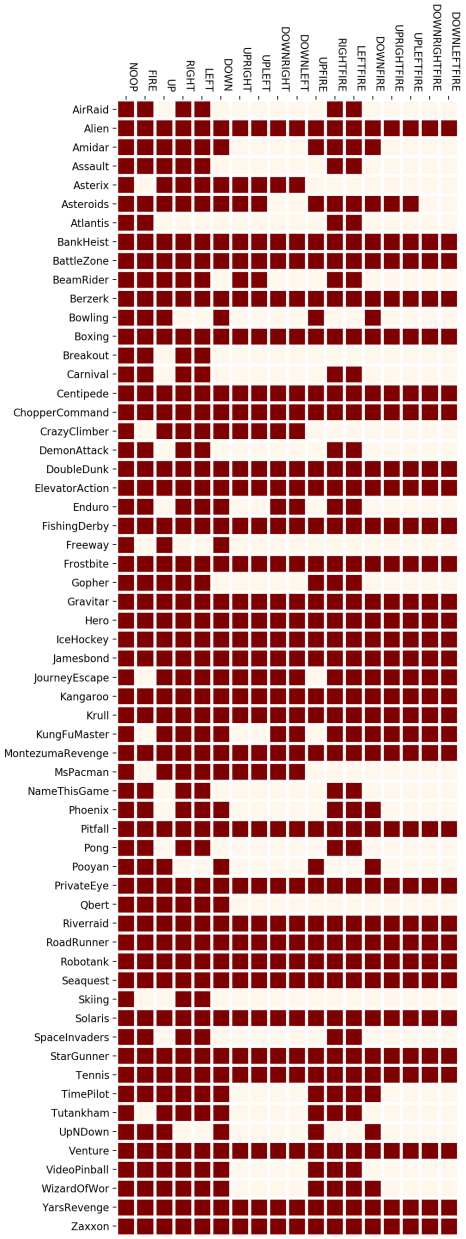
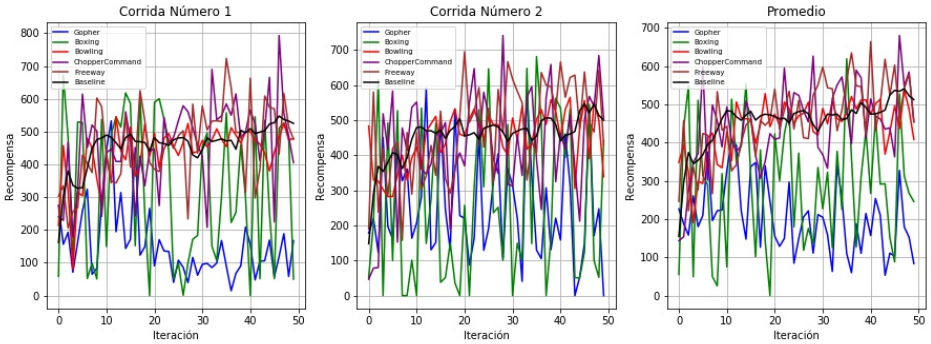


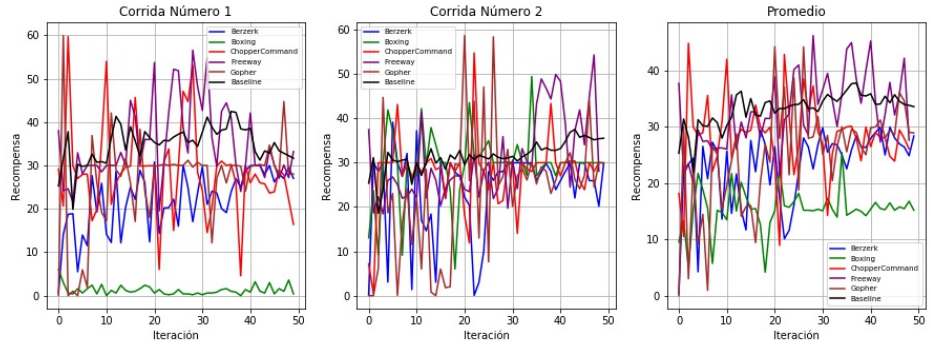
Figura 20: Acciones disponibles para cada juego utilizado en los experimentos de la biblioteca Dopamine [Castro et al., 2018].

B Curvas de Aprendizaje

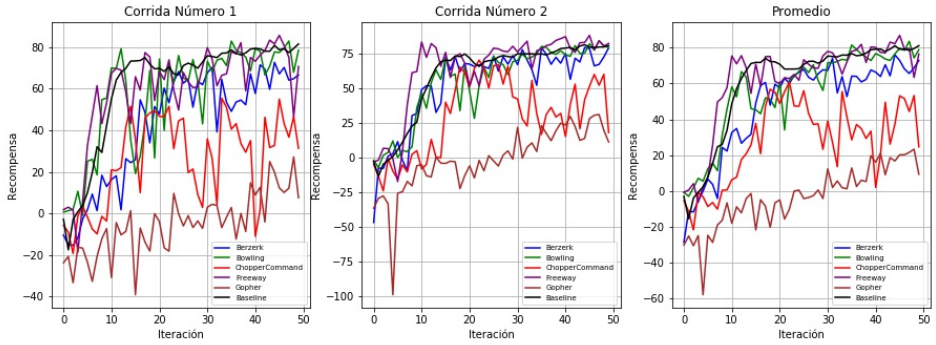
En esta sección se presentan las curvas de aprendizaje completas asociadas a los experimentos de la sección 5, en la Figura 21 se muestran las dos corridas independientes junto con el promedio de ambas. Mientras que en la Figura 22 se muestran cinco corridas por cada juego seleccionado en color morado se pueden observar las curvas que se utilizan como *baseline* de los experimentos mostrados en la sección 5.



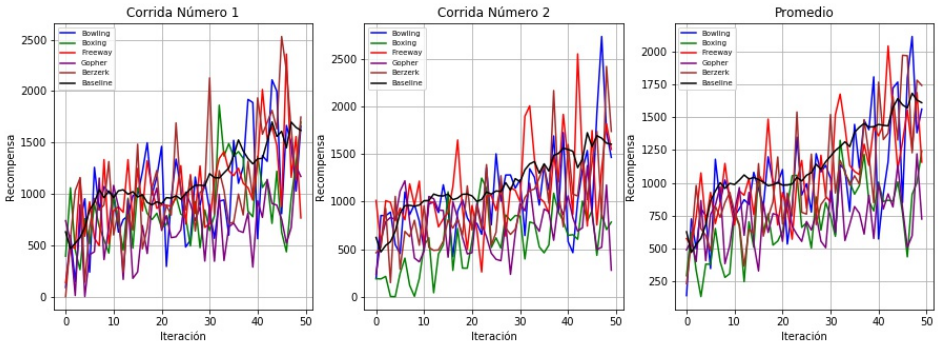
(a) Berzerk



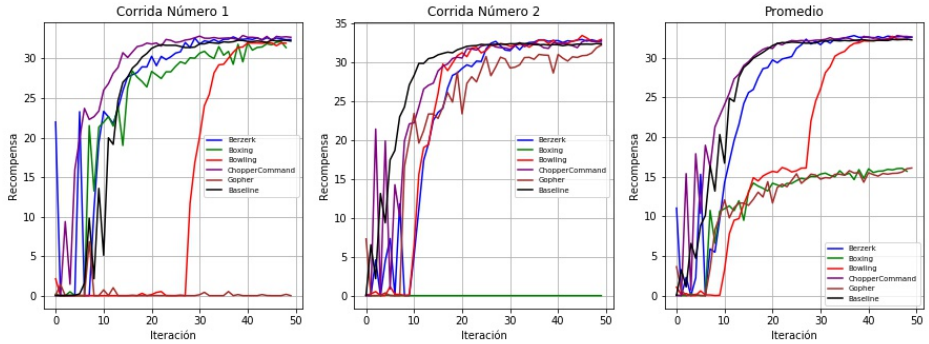
(b) Bowling



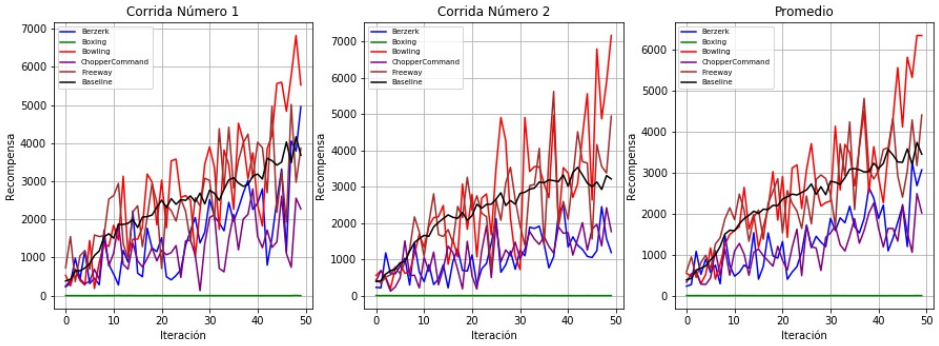
(c) Boxing



(d) Chopper Command

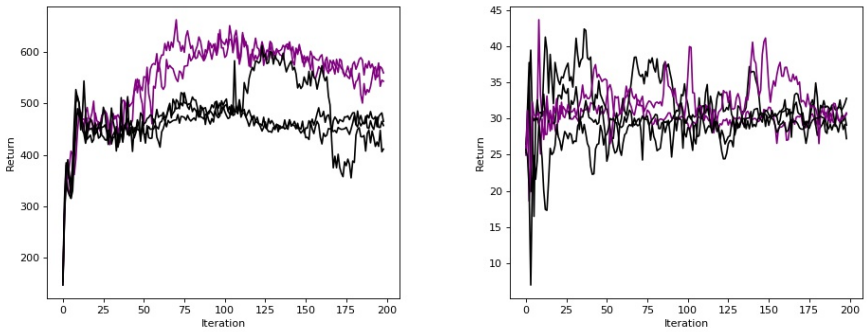


(e) Freeway



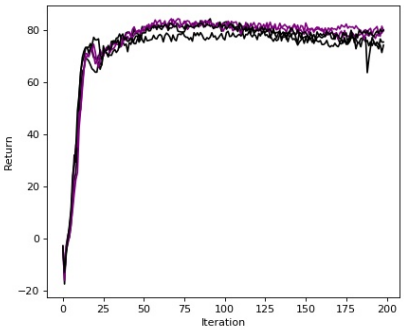
(f) Gopher

Figura 21: Curvas de aprendizaje con 50 iteraciones.

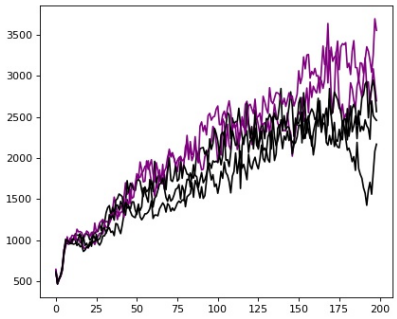


(a) Berzerk

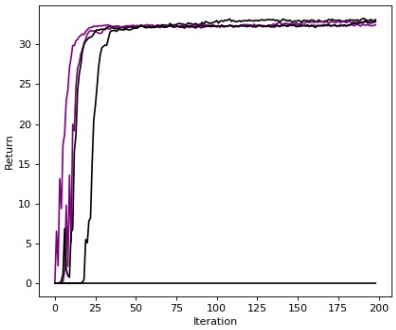
(b) Bowling



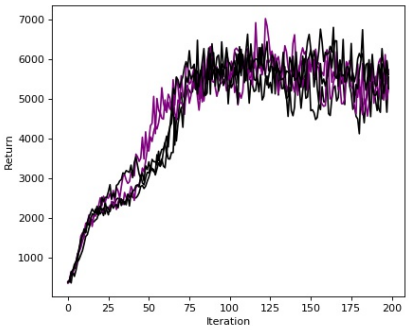
(c) Boxing



(d) Chopper Command



(e) Freeway



(f) Gopher

Figura 22: Curvas de aprendizaje proporcionadas por la biblioteca Dopamine-RL [Castro et al., 2018]

Bibliografía

[TPU,] Cloud tensor processing unit (tpu). https://cloud.google.com/tpu/docs/tpus?hl=es-419. Accessed: 2019-07-07.

[Bellemare et al., 2017] Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887.

[Bellemare et al., 2013] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

[Burger et al., 2009] Burger, W., Burge, M. J., Burge, M. J., and Burge, M. J. (2009). *Principles of digital image processing*. Springer.

[Carr et al., 2018] Carr, T., Chli, M., and Vogiatzis, G. (2018). Domain adaptation for reinforcement learning on the atari. *arXiv preprint arXiv:1812.07452*.

[Castro et al., 2018] Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. (2018). Dopamine: A Research Framework for Deep Reinforcement Learning.

[Cheng et al., 2017] Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[Dabney et al., 2018a] Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018a). Implicit quantile networks for distributional reinforcement learning. *CoRR*, abs/1806.06923.

[Dabney et al., 2018b] Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018b). Implicit quantile networks for distributional reinforcement learning. *arXiv preprint arXiv:1806.06923*.

[de la Cruz et al., 2016] de la Cruz, G., Du, Y., Irwin, J., and Taylor, M. (2016). Initial progress in transfer for deep reinforcement learning algorithms.

[Fawaz et al., 2018] Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., and Muller, P.-A. (2018). Transfer learning for time series classification. *arXiv preprint arXiv:1811.01533*.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Güler et al., 2018] Güler, R. A., Neverova, N., and Kokkinos, I. (2018). Densepose: Dense human pose estimation in the wild. *arXiv preprint arXiv:1802.00434*.

[Hessel et al., 2017] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298.

[Kaiser et al., 2019] Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Sepassi, R., Tucker, G., and Michalewski, H. (2019). Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374.

[Kouw, 2018] Kouw, W. M. (2018). An introduction to domain adaptation and transfer learning. *arXiv preprint arXiv:1812.11806*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Learned-Miller,] Learned-Miller, E. Cloud tensor processing unit (tpu).

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

[Machado et al., 2018] Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562.

[Mitchell et al., 1997] Mitchell, T. M. et al. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877.

[Mittel et al., 2018] Mittel, A., Munukutla, S., and Yadav, H. (2018). Visual transfer between atari games using competitive reinforcement learning. *arXiv preprint arXiv:1809.00397*.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[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.

[Parisotto et al., 2016] Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2016). Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*.

[Rusu et al., 2015] Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2015). Policy distillation. *arXiv preprint arXiv:1511.06295*.

[Rusu et al., 2016] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

[Schaul et al., 2015] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *CoRR*, abs/1511.05952.

[Schmitt et al., 2018] Schmitt, S., Hudson, J. J., Zidek, A., Osindero, S., Doersch, C., Czarnecki, W. M., Leibo, J. Z., Kuttler, H., Zisserman, A., Simonyan, K., et al. (2018). Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

[Silver et al., 2017] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354.

[Skansi, 2018] Skansi, S. (2018). *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[Sutton, 2018] Sutton, B. (2018). *Reinforcement Learning: an Introduction*.

[Taylor and Stone, 2009] Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

[van Hasselt et al., 2015] van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461.

[Wang and Deng, 2018] Wang, M. and Deng, W. (2018). Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153.

[Wang et al., 2015] Wang, Z., de Freitas, N., and Lanctot, M. (2015). Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581.

[Weiss et al., 2016] Weiss, K., Khoshgoftaar, T. M., and Wang, D. (2016). A survey of transfer learning. *Journal of Big Data*, 3(1):9.

[Yin and Pan, 2017] Yin, H. and Pan, S. J. (2017). Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In *AAAI*, pages 1640–1646.

[Yosinski et al., 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

[Zahavy et al., 2017] Zahavy, T., Mankowitz, D. J., and Mannor, S. (2017). Deep reinforcement learning from expert's experience replay.

[Zhuang et al., 2015] Zhuang, F., Cheng, X., Luo, P., Pan, S. J., and He, Q. (2015). Supervised representation learning: Transfer learning with deep autoencoders. In *IJCAI*, pages 4119–4125.