Universidad Zaragoza

Trabajo Fin de Grado

Enseñando al ordenador a jugar a videojuegos mediante aprendizaje profundo por refuerzo

Teaching a computer how to play videogames using Deep Reinforcement Learning

Autor

Alberto Sabater Bailón

Directores

Carlos Bobed Lisbona
Eduardo Mena Nieto

Grado en Ingeniería Informática
Escuela de Ingeniería y Arquitectura
Junio de 2016

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Alberto Sabater Bailón,

con nº de DNI 25202202K en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Enseñando al ordenador a jugar a videojuegos mediante aprendizaje profundo por refuerzo

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 24 de Junio de 2016

Fdo: Alberto Sabater Bailón

Enseñando al ordenador a jugar a videojuegos mediante aprendizaje profundo por refuerzo

Resumen

Uno de los mayores aspectos a considerar a la hora de trabajar con algoritmos de Aprendizaje Automático es la relación existente entre la arquitectura de la red neuronal, la complejidad de los datos de entrenamiento, el tiempo invertido en el aprendizaje y la calidad de los resultados obtenidos.

El problema que aquí se aborda es el entrenamiento de un agente para que sea capaz de jugar a videojuegos. Debido a su complejidad, este problema es tratado con grandes modelos de redes neuronales como la utilizada por el grupo Google DeepMind en el proyecto del que parte este trabajo. En él, se entrena una red neuronal profunda mediante aprendizaje por refuerzo para que aprenda a jugar a juegos de Atari 2600. Dicho entrenamiento aprende la acción óptima a realizar en cada situación tomando como entrada la pantalla de juego y la puntuación conseguida en cada momento. Se ha llevado a cabo un análisis exhaustivo de todo el proceso de aprendizaje así como de los resultados obtenidos por el mismo, con el fin de identificar posibles alternativas que conduzcan a una mejora de los resultados y/o a una mejora de la velocidad de convergencia del algoritmo.

Como resultado de este análisis, se ha diseñado una nueva arquitectura de la red neuronal en la que se han usado pesos pre-entrenados. Esta inicialización de la red se ha llevado a cabo mediante la trasferencia de conocimiento de otros modelos entrenados con juegos de características similares y mediante el aprendizaje de características de forma no supervisada. Para este último caso, se ha llevado a cabo un estudio de diferentes metodologías de entrenamiento y se ha probado finalmente la eficiencia de la generación de pesos mediante K-means y autocodificadores. Como entrada de este aprendizaje no supervisado se han utilizado vídeos de diferentes partidas subidas por la comunidad a YouTube, de los que se han extraído y adaptado los frames que los componen.

Agradecimientos

Agradecer en primer lugar a mis directores Carlos Bobed y Eduardo Mena, por toda la ayuda y apoyo recibidos durante la realización de este proyecto.

A todos aquellos informáticos que he conocido durante la carrera, con los que he compartido momentos de estrés y alegría, y que han hecho de este caos cuatro años inolvidables.

A los de siempre, *los chipis*, que de una u otra manera sin ellos todo esto no hubiera sido posible. Por todos los momentos compartidos y por los que están por venir.

Finalmente, dar las gracias a mi familia. A mis padres y a mi hermana, por su inestimable apoyo durante todos estos años.

Índice

1. Introducción

Inspirados en el sistema nervioso biológico, las redes de neuronales artificiales están formadas por un gran número de unidades de procesamiento (neuronas) interconectadas entre sí siguiendo una determinada arquitectura. Los numerosos avances en la investigación de sistemas inteligentes y las tecnologías actuales, han permitido utilizar redes neuronales artificiales en multitud de tareas como la visión por computador, optimización de tareas o análisis de datos.

El aumento de la capacidad de computación que se ha generado en los últimos años ha permitido el uso de modelos más complejos de aprendizaje automático conocidos como aprendizaje profundo (*Deep Learning*). Estos modelos están formados por numerosas capas de neuronas, que consiguen aprender abstracciones no lineales de alto nivel a partir de datos de grandes dimensiones.

Una de las aplicaciones de las redes neuronales es la simulación del comportamiento humano mediante aprendizaje por refuerzo (*Reinforcement Learning*). Esta técnica se enfrenta a problemas decisionales en los que un agente debe tomar la acción o conjunto de acciones que optimicen una determinada tarea. El problema que aborda este proyecto es el uso de este tipo de aprendizaje en un agente que juegue a videojuegos basándose únicamente en la pantalla y la puntuación obtenida de los mismos.

Para facilitar el proceso de aprendizaje y la simulación del comportamiento humano, los datos de entrenamiento utilizados provienen de videojuegos de Atari 2600, una consola cuyas características relativas a la libertad de acciones y gráficos son más limitadas que otras de su género. No obstante, debido a la dimensionalidad de la información generada por estos juegos, ésta es tratada con redes neuronales profundas (*Deep Neural Networks*), cuya misión es realizar un análisis de la pantalla de juego y en consecuencia, elegir la acción que optimice la puntuación obtenida en dicho juego.

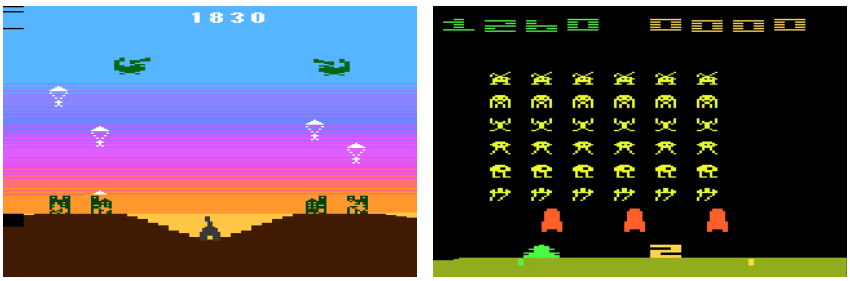


Figura 1: Ejemplos de juegos de Atari 2600. A la izquierda *Commando Raid*. A la derecha *Space Invaders*.

A pesar de su gran potencial, las redes neuronales profundas, son muy difíciles de entrenar debido a la gran cantidad de parámetros con los que trabajan [1, 2]. De entre las diferentes técnicas que se han utilizado para reducir la dimensionalidad de los datos de entrenamiento , las redes convolucionales (Convolutional Networks) son la referencia en el estado del arte para tareas como el reconocimiento de imágenes y vídeo o el procesamiento del lenguaje natural. De la fusión de estas dos arquitecturas nace un nuevo modelo conocido como red convolucional profunda (Deep Convolutional Network).

La existencia de tantos parámetros entrenables puede conducir a situaciones no deseadas como el sobreajuste, convergencia a mínimos locales o un largo proceso de entrenamiento. Para reducir estos riesgos, se han desarrollado diferentes técnicas de aprendizaje no supervisado que ayudan a predecir patrones entre datos no etiquetados. Los resultados obtenidos de este proceso se toman como punto de partida del aprendizaje principal. Otra técnica utilizada en la inicialización un entrenamiento, es la 'trasferencia de conocimiento', que aprovecha los patrones aprendidos de una red neuronal para pre-entrenar una nueva.

1.1. Objetivo y alcance

El objetivo de este proyecto es el estudio y aplicación de técnicas de aprendizaje automático a la toma de decisiones de un agente que juega a videojuegos clásicos. Para ello, replicando el proyecto de Google DeepMind [3], que utiliza técnicas de aprendizaje por refuerzo aplicadas a juegos de Atari 2600, se busca obtener un mayor conocimiento de la teoría subyacente a su entrenamiento y analizar aquellos factores que influyen en la calidad del aprendizaje.

Una vez identificados estos factores, se va a realizar un estudio de las diferentes técnicas que pueden contribuir a una mejora potencial del sistema y su actual estado del arte. Como resultado de este estudio se van a seleccionar e implementar aquellas metodologías de aprendizaje automático que buscan una mejora del rendimiento o de los resultados obtenidos en el proceso de aprendizaje. Tras esta implementación se va a diseñar un banco de pruebas acorde con los recursos disponibles con el fin de poder evaluar la eficiencia de las técnicas desarrolladas. Del análisis de los resultados obtenidos, se van a proponer nuevas líneas de investigación para una continuación del trabajo desarrollado.

1.2. Contenidos de la memoria

En la Sección 2 se describen los conceptos teóricos y metodologías de aprendizaje automático estudiadas que forman el contexto tecnológico, y en la siguiente sección (Sección 3) se describen las tecnologías y herramientas que han sido utilizadas en el desarrollo de este proyecto.

La Sección 4 contempla el funcionamiento del proceso de aprendizaje desarrollado por DeepMind y se incluye una descripción de las modificaciones reali-

zadas para su adaptación a este trabajo.

La Sección 5 describe los detalles de la implementación de los diferentes algoritmos de aprendizaje no supervisado desarrollados, y la Sección 6 reune el conjunto de pruebas realizadas y un análisis de los resultados obtenidos de éstas.

Finalmente, en la Sección 7 se exponen las conclusiones obtenidas del desarrollo de este proyecto junto con el cronograma seguido, y se proponen nuevas líneas de investigación para la continuación de este trabajo.

2. Contexto tecnológico

A continuación se explican los conceptos teóricos cuyo estudio ha sido imprescindible para el desarrollo de este proyecto. Esta sección comienza explicando el funcionamiento de las redes neuronales, para profundizar después en el detalle de las redes neuronales convolucionales. Posteriormente se describen los fundamentos del aprendizaje por refuerzo y se acaba con una explicación del aprendizaje de características y sus diferentes formas de entrenamiento supervisado y no supervisado.

2.1. Redes neuronales

Una neurona artificial [4] es una unidad de cálculo que toma como entrada un conjunto de valores que procesa para producir una determinada salida. Esta salida es generada de acuerdo con la siguiente fórmula

$$y = f\left(\sum_{i=1}^{n} w_i x_i - b\right), \tag{1}$$

en la que se calcula el producto escalar entre la entrada $X = \{x_1, x_2, ..., x_n\}$ y los pesos $W = \{w_1, w_2, ..., w_n\}$ de la neurona. A este resultado se le aplica un sesgo o bias b, y el valor obtenido es procesado por una función de activación f, de la que se obtiene la salida de la neurona y. Los parámetros de esta unidad (pesos y bias) son modificados durante el proceso de aprendizaje para calcular la salida que optimice la tarea que le es asignada.

Una red neuronal es un modelo formado por un conjunto de neuronas conectadas entre sí, que trabajan juntas para optimizar una determinada tarea. El modelo de red neuronal más sencillo es el perceptrón multicapa, con una organización en capas en el que las neuronas de una capa únicamente están conectadas con las neuronas de la siguiente.

El aprendizaje de una red neuronal se puede clasificar en tres grandes categorías en función de los datos de entrenamiento utilizados. Los datos etiquetados son utilizados en el aprendizaje supervisado para predecir una determinada salida conocida. El entrenamiento por refuerzo es utilizado cuando no se conoce la salida que debería tener la red neuronal, pero su calidad se puede valorar en función de la realimentación que se obtiene del entorno. Cuando no se tiene ningún tipo de estimación que guíe el aprendizaje, se lleva a cabo el aprendizaje no supervisado, que estudia la estructura interna de los datos de entrada.

2.1.1. Redes neuronales convolucionales

Las redes neuronales convolucionales (*Convolutional Neural Network*, CNN), son una variación de los perceptrones multicapa. Inspiradas en el funcionamiento del cortex visual, las capas convolucionales están formadas por neuronas que

asumen una correlación espacial de los datos de entrada. El uso de neuronas convolucionales, reduce el número de parámetros entrenables y mejora la eficiencia del entrenamiento frente al uso de neuronas totalmente conectadas.

La aplicación de redes neuronales convolucionales ha sido ampliamente usada desde su introducción [5], consiguiendo un alto rendimiento en tareas como la visión por computador y el procesamiento de lenguaje natural.

Asumiendo una correlación espacial en los datos de entrada, cada neurona mantiene una conectividad local con las neuronas de su capa anterior, es decir, únicamente se comunica con un subconjunto de las neuronas de la capa que le precede, al que se le denomina campo receptivo (*receptive field*), como se muestra en la Figura 2. Además, los parámetros (*filtro* ó *feature*) que se aplican sobre el campo receptivo son iguales para todas las neuronas que forman una capa convolucional.

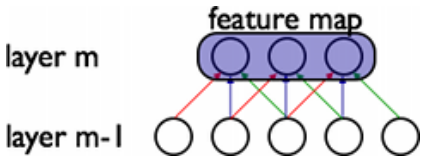


Figura 2: *Feature map* generado por un filtro de dimensión 3x1. Los pesos del mismo color tienen el mismo valor[1].

La salida de una capa convolucional se genera aplicando el producto escalar entre el filtro de dicha capa y cada uno de los campos receptivos de la capa que le precede. Esta *convolución* produce un resultado de una dimensionalidad menor a la que se le denomina *feature map*. Para obtener una representación más variada de los datos de entrada, cada capa convolucional oculta está formada por múltiples feature maps, obtenidos por la convolución de múltiples filtros sobre los datos de entrada de la capa.

El resultado obtenido por la convolución de los filtros sobre los datos de entrada depende de los siguientes parámetros:

Paso (*stride*): indica la separación entre los centros de cada campo receptivo, es decir el nivel de solapamiento entre ellos.

Relleno (*padding*): al desplazar el centro del campo receptivo hacía los bordes de los datos de entrada, parte de este campo puede quedar en una zona sin información. El relleno indica que cantidad del campo receptivo puede quedar vacía y por lo tanto rellenada con ceros.

Tamaño de filtro: indica el número de neuronas que componen un filtro.

[1] `http://deeplearning.net/tutorial/lenet.html`

Profundidad: indica el número de filtros que componen una capa convolucional.

La arquitectura más común de una red neuronal convolucional está formada por el uso de un rectificador lineal ReLU y opcionalmente, una capa de agrupación *pooling layer* después de cada capa convolucional. Adicionalmente otras arquitecturas hacen uso de capas de dropout, dropout espacial o capas convolucionales totalmente conectadas.

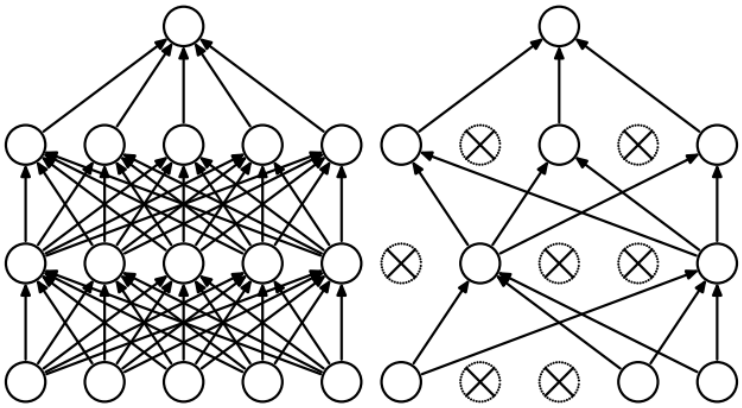
ReLU: es una capa de neuronas que aplican la función de activación máx$(0, x)$ a sus datos de entrada, generalmente producidos por una capa convolucional o totalmente conectada.

Capa de agrupación: aplica una ventana sobre la información generada por una capa convolucional, y para cada una de estas sub-regiones, únicamente genera un valor en su salida. El uso de esta técnica contribuye a reducir la dimensionalidad de los datos con los que se trabaja y con ello, la necesidad de computación para las capas de la red neuronal que le siguen. Este tipo de capas también proporcionan invariancia a la traslación en la convolución de los filtros, una característica deseada en el uso de redes neuronales convolucionales para tareas como la clasificación de imágenes. La capa de agrupación más común es el denominado max-pooling, que toma como salida el valor más alto en cada una de las sub-regiones de sus datos de entrada. Otros tipos de capas de agrupación son la agrupación promedio (*averale pooling*) y *L2-norm pooling*.

Dropout [6]: desactiva cada neurona de la capa sobre la que actúa con una probabilidad p en cada iteración del algoritmo de aprendizaje. Cada neurona desactivada no es tenida en cuenta por la siguiente capa de neuronas, ni se calcula su gradiente respecto a la función de coste. Dropout contribuye a que cada neurona aprenda de forma independiente al resto evitando el sobreajute de la red neuronal. Una vez acabado el entrenamiento, durante la evaluación del mismo no se desactiva ninguna neurona.

Dropout espacial [7]: actúa de forma similar al dropout pero tiene en cuenta la correlación espacial de las neuronas de una capa convolucional. De esta forma en lugar de desactivar cada neurona de forma individual, desactiva el conjunto de neuronas que forman un filtro de la capa convolucional sobre la que actúa.

Capa convolucional totalmente conectada [8]: toma como entrada una matriz multidimensional y aumenta su tamaño, infiriendo el resultado, mediante la convolución de filtros, de forma análoga a una capa convolucional.



(a) Red neuronal normal (b) Red neuronal con dropout

Figura 3: Aplicación de dropout sobre una red neuronal[2]

Tras las capas explicadas anteriormente, es frecuente el uso de capas totalmente conectadas, como se puede ver en la Figura 4, en las que cada neurona de esta capa se conecta con todas las neuronas de la capa que le precede.

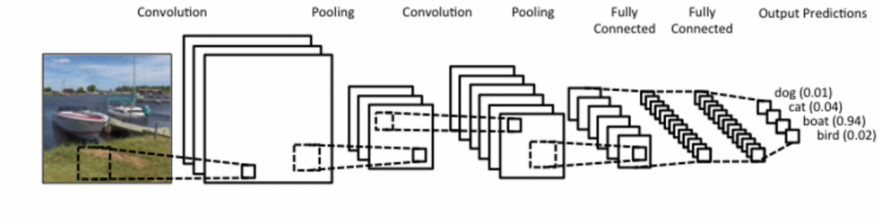


Figura 4: Arquitectura de una red convolucional formada por capas convolucionales, max-pooling y capas totalmente conectadas[3].

2.2. Aprendizaje por refuerzo

El aprendizaje por refuerzo es un tipo de aprendizaje automático inspirado en la psicología conductista, que estudia el aprendizaje de un individuo mediante refuerzo y castigo en la tarea de optimizar su comportamiento con el entorno que le rodea. El uso de este tipo de algoritmos ha sido ampliamente utilizado en disciplinas como la teoría de juegos, robótica o planificación de tareas.

[2]Dropout: A Simple Way to Prevent Neural Networks from Overfitting [6]
[3]https://www.clarifai.com/technology

Los algoritmos de aprendizaje por refuerzo buscan la solución de un Proceso de Decisión de Markov (*Markov Decision Problem*, MDP). Un MDP parte de un modelo basado en cadenas de Markov [9], creando un problema que consta de un conjunto de estados, acciones, probabilidades de transición de un estado a otro, recompensas de las transiciones, una política de acciones y una métrica del rendimiento. El objetivo del problema es encontrar la política que maximice la métrica de rendimiento. Las métricas de rendimiento más comunes son la media de las recompensas obtenidas (*average reward*) y la suma de las recompensas obtenidas, ponderadas por un factor que modifica el valor de cada recompensa en función del momento en la que se obtuvo (*discounted reward*).

La metodología de aprendizaje por refuerzo que se utiliza en este proyecto es el Q-learning [10], que en lugar de buscar la valoración única de un estado, busca el cálculo del llamado Q-factor para cada par estado-accion, Q(s,a). Este valor indica la máxima recompensa obtenida eligiendo la acción a en el estado s. Este Q-factor es estimado mediante una red neuronal profunda (*Deep Q-network*, DQN), de forma que una vez que la red esté entrenada, esta será capaz de deducir las recompensas futuras de cada acción dado un estado, siendo aquella que maximice este valor la mejor acción a elegir. La obtención de estos valores se calcula con la función de recompensa descontada (*discounted reward*), que pondera las recompensas futuras por un factor γ (2).

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi], \tag{2}$$

donde r_t es la recompensa en cada instante de tiempo t ponderada por γ, alcanzable siguiendo la política $\pi = P(a|s)$ después de realizar la acción a sobre el estado s.

El aprendizaje de esta DQN se realiza por un descenso de gradiente estocástico [11] (*stochastic gradient descent*), en el que la recompensa inmediata se utiliza como realimentación para actualizar la red neuronal de forma que los valores Q(s,a) estimados sean más próximos a su valor real. Para conseguirlo, el objetivo del aprendizaje es minimizar el resultado de la función de coste (3). Cada *experiencia* (estado actual y siguiente, acción elegida y recompensa conseguida) es almacenada en un dataset D, de forma que posteriormente se puedan aplicar las actualizaciones de los pesos de la red, sobre lotes (*minibatches*) del conjunto de experiencias almacenadas. Estos lotes son seleccionados de forma aleatoria sobre el conjunto total de datos guardados.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a', \theta_i^-) - Q(s, a; \theta_i)\right)^2\right], \tag{3}$$

donde γ es el factor de descuento, θ_i son los pesos de la Q-network en la iteración i y θ_i^- son los pesos de la red utilizados para el cálculo de los valores objetivo de la actualización de los pesos en la iteración i del descenso de gradiente.

Para más detalles sobre el uso de DQNs en aprendizaje por refuerzo, referimos al lector a los papers escritos por DeepMind en arXiv [12] y Nature [3].

2.3. Aprendizaje de características

Una característica (*feature*) es una propiedad de los datos utilizados para el entrenamiento de una red neuronal, que se aprende en el trascurso de éste y es necesaria para llevar a cabo la tarea que le ha sido asignada a la red. En su aplicación al campo de la visión por computador, una característica aporta propiedades como la detección de ejes, esquinas y otras formas de las que se compone la imagen. En una red convolucional, estas características equivaldrían a los pesos que componen sus filtros.

Para maximizar la calidad de un entrenamiento, es necesario que las características aprendidas sean lo más dispersas posible entre sí, de forma que cada una de ellas pueda aportar información distinta al aprendizaje. Para facilitar facilitar esta dispersión, una buena inicialización de los pesos de la red neuronal es fundamental. Se ha demostrado que una inicialización de los pesos de forma aleatoria siguiendo una determinada distribución de probabilidad, comúnmente una distribución gaussiana entorno a cero [13], aporta un gran beneficio al aprendizaje. Sin embargo, un aprendizaje previo de la estructura de los datos puede ser utilizado en la creación de estas características iniciales, que durante el aprendizaje principal podrán ser modificadas para adaptarse a las necesidades de la red.

El uso de estas características ya aprendidas como inicialización de la red neuronal, contribuye a una reducción del tiempo de entrenamiento y del sobreajuste que se puede producir por la complejidad del modelo o de los datos de entrenamiento [14]. A continuación se describen diferentes técnicas de aprendizaje de características de forma supervisada y no supervisada.

2.3.1. Aprendizaje no supervisado de características

El objetivo principal del entrenamiento no supervisado, es la búsqueda de patrones en conjuntos de datos no etiquetados y aparentemente no estructurados. Las aplicaciones más comunes del aprendizaje no supervisado son el clústering y la reducción de la dimensionalidad de los datos de entrenamiento.

Para el caso que nos ocupa, se ha realizado un estudio de diferentes metodologías del área del aprendizaje no supervisado [15, 14] para la extracción de características que sirvan como pesos pre-entrenados a las capas convolucionales de la red neuronal convolucional que se entrena mediante aprendizaje por refuerzo.

A continuación se describen las principales técnicas de aprendizaje de características mediante entrenamiento no supervisado analizadas que han tenido una implicación en este proyecto, y en el Anexo A, se describe otra de las técnicas analizadas más importante, cuya aplicación a este proyecto no ha sido posible.

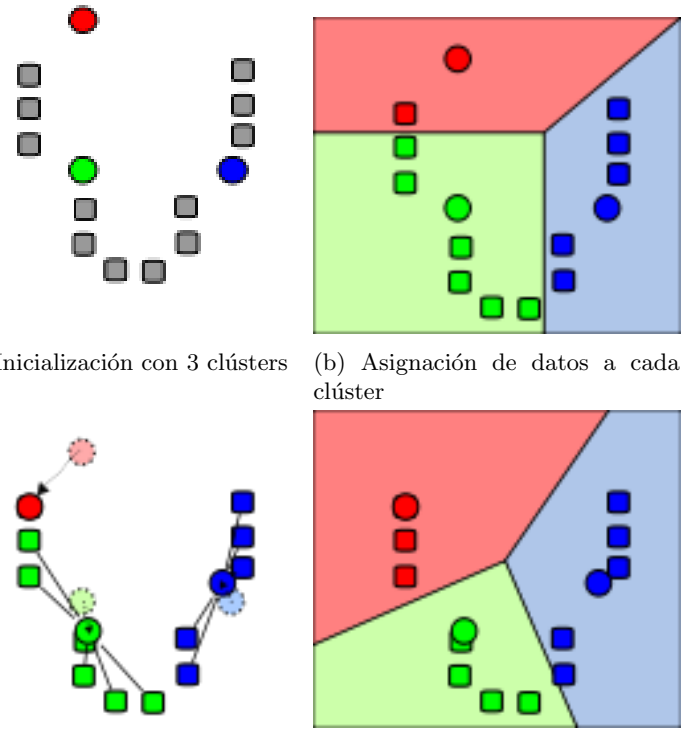
K-means clústering

El objetivo del K-means clústering [16], es el de clasificar un conjunto de datos agrupándolos en k clústers. Éste es uno de los algoritmos de aprendizaje no supervisado más simples para el problema de clusterización y requiere poca capacidad de computación, por lo que es capaz de procesar gran cantidad de información en poco tiempo. Es por ello, por lo que ha sido utilizado en tareas como la segmentación de mercado, geoestadística y astronomía.

La implementación más común para k-means comienza estableciendo el centroide de cada clúster de forma aleatoria dentro del conjunto de datos de entrada. En cada iteración del algoritmo, cada dato de entrada se asigna al clúster cuyo centroide es el más cercano y se recalcula la ubicación de los centroides para que vuelvan a estar en el centro de su clúster. Estas dos acciones se repiten hasta que los centroides quedan en una posición fija, entonces se puede decir que el algoritmo ha convergido. Generalmente la función elegida para medir la distancia entre un dato y un centroide, cuyo resultado se quiere minimizar, es el error cuadrático medio (*Mean Squared Error*) (4). Un ejemplo de ejecución de K-means se puede ver en la Figura 5.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\widehat{Y}_i - Y_i\right)^2 \tag{4}$$

La calidad de los resultados obtenidos por K-means depende de la inicialización del algoritmo ya que puede converger a mínimos locales, por lo que es común probarlo varias veces con centroides aleatorios. Si el número de clústers se puede deducir a priori, es recomendable usar esta información para inicializar k, de lo contrario, también se puede deducir mediante cross-validation [17]. Ésta técnica se basa en la repetición de un entrenamiento con diferentes parámetros de aprendizaje para definir cuales son los que optimizan su resultado.



(a) Inicialización con 3 clústers

(b) Asignación de datos a cada clúster

(c) Actualización de los centroides

(d) Asignación de datos a los clústers actualizados

Figura 5: Ejemplo de ejecución de K-means[4]

Otra de las situaciones no deseadas a las que se puede enfrentar el algoritmo de K-means es la genearación de clústers vacíos, es decir, que alguno de los clústers obtenidos tras la convergencia no tenga ningún dato asignado. Para reducir la probabilidad de encontrarnos frente a esta situación, es necesaria una buena inicialización de los centroides. No obstante, también se pueden descartar los clústers vacíos ya que el resto de centroides aportan toda la información necesaria.

En el contexto de este proyecto, se utiliza K-means para el aprendizaje de los pesos que formarían cada uno de los filtros iniciales de una capa convolucional en una red neuronal convolucional [18, 19]. De esta forma, cada uno de estos filtros está representado por un centroide con su misma dimensionalidad. Esta dimensionalidad también es compartida por los datos de entrenamiento, que corresponden a subregiones de las imágenes de entrenamiento. Una vez que el algoritmo ha convergido, las características aprendidas son lo suficientemente

[4] `https://en.wikipedia.org/wiki/K-means_clustering`

dispersas como para poder generar una representación de los datos de entrada útiles para el aprendizaje, sirviendo como pre-entrenamiento de éste.

A pesar de su sencillez, K-means ha demostrado obtener un rendimiento similar al de otras técnicas de aprendizaje de características más sofisticadas [19], como las descritas a continuación, llegando incluso a superarlas [14].

Autocodificadores

Un autocodificador (*autoencoder*) [20] es una red neuronal cuyo objetivo es obtener una dimensionalidad menor de datos de entrada no etiquetados [21]. Para ello trata de reconstruir la entrada de la red a partir de la nueva representación de los datos generada.

Un autocodificador está formado por dos partes, el codificador (*encoder*), que genera una representación de los datos de entrada con una dimensionalidad menor; y el decodificador (*decoder*), que trata de reconstruir la entrada de la red a partir de esta representación. Ambas partes pueden estar formadas por más de una capa de neuronas, con el fin de obtener una mejor representación de los datos de entrenamiento. De forma que cuánto más interna es una capa del autocodificador, menor dimensionalidad debe tener. Lo contrario ocurre con el decodificador, cuya última capa tiene la misma dimensionalidad que los datos de entrada de la red. Comúnmente, cada capa neuronal del codificador tiene una capa análoga en el decodificador. Adicionalmente, se puede hacer uso de una función de activación al comienzo del decodificador, como la tangente hiperbólica, para mejorar el rendimiento del entrenamiento. La estructura de un autocodificador se puede ver en la Figura 6.

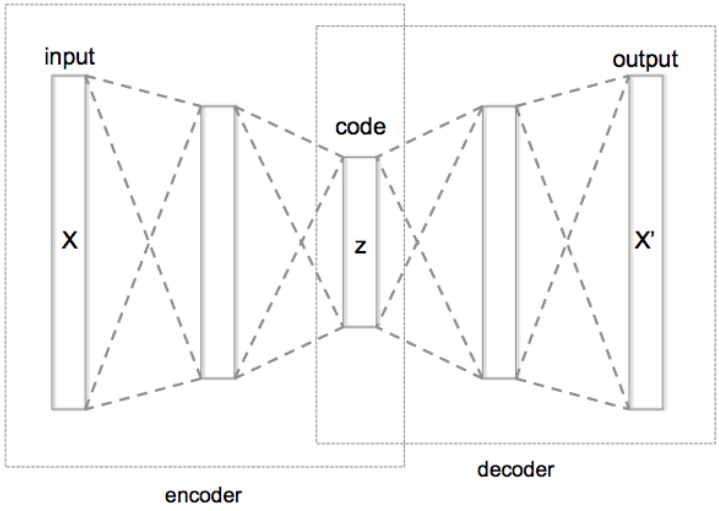


Figura 6: Arquitectura de un autocodificador[5].

Esta red neuronal se entrena mediante un descenso de gradiente que trata de minimizar la función de error cuadrático medio (4), en la que la predicción $\widehat{Y}_i$ es la salida de la red neuronal, e Y corresponde a la entrada de la red neuronal, cuyo valor se quiere alcanzar.

Una vez que la red está entrenada, las capas de neuronas (pesos y bias) del codificador han aprendido una serie de características lo suficientemente dispersas como para obtener una representación de los datos de entrada, con una dimensionalidad menor, capaz de obtener una reconstrucción aproximada de la entrada de la red neuronal. Como el codificador ha aprendido la correlación existente de los datos de entrada, estas características pueden ser usadas como pesos pre-entrenados de una nueva red neuronal que utilice los mismos datos de entrenamiento.

Para aprender las filtros de una red convolucional se hace uso de un autocodificador convolucional [22], en el que las capas de neuronas que forman su codificador son capas convolucionales. De esta forma, al igual que en las redes convolucionales, las neuronas buscan una correlación espacial de los datos de entrada de la red.

Para poder reconstruir los datos de entrada mediante la representación aprendida por el codificador, las capas que forman el decodificador, son capas totalmente convolucionales, por lo que aumentan la dimensionalidad de la representación obtenida suponiendo una correlación espacial de los valores que la componen. Para invertir la reducción de dimensionalidad de las capas de agrupamiento de una red convolucional, se hace uso de capas de 'des-agrupamiento' que aumentan la dimensionalidad mediante el uso de heurísticas.

2.3.2. Aprendizaje supervisado: transferencia de conocimiento

La transferencia de conocimiento es uno de los principales recursos del aprendizaje humano. El aprendizaje de una tarea tiende a ser más rápido y eficiente cuando ya se ha trabajado en una situación similar. Por ejemplo, el reconocimiento de un nuevo animal, no requiere la visualización de múltiples muestras del mismo, ya que se parte del conocimiento previo de otros muchos animales y objetos. Lo mismo pasaría en el caso que nos ocupa, cuando una persona tiene mucha experiencia jugando a videojuegos, el aprendizaje de uno nuevo será mucho más rápido que el de una persona que no cuenta con esta experiencia. De forma análoga, el conocimiento aprendido en redes de neuronas artificiales puede ser aprovechado para el aprendizaje de una nueva tarea de características similares [23].

El estudio de redes neuronales profundas entrenadas a partir de imágenes, revela que las características aprendidas por estas capas aportan información más

[5] `https://en.wikipedia.org/wiki/Autoencoder`

general a un entrenamiento que las capas más profundas [24, 25]. Las representaciones generadas por las capas superficiales tienden a mostrar información de formas de la imagen como ejes o esquinas. Estas características generales no son específicas para determinados datasets o tareas, por lo que redes entrenadas para diferentes tareas pueden contener características similares en sus primeras capas. Por el contrario, las capas más profundas tienden a aprender conceptos más específicos para la tarea y dataset que le son asignados.

La mejora del rendimiento provocada por la transferencia de conocimiento entre dos redes neuronales decrece cuanto más diferentes son los objetivos de ambos modelos o cuando se transfieren características muy específicas, sin embargo su uso en el peor caso es preferible a una inicialización aleatoria de los pesos de la red.

La metodología más común en la transferencia de conocimiento consiste en clonar los pesos y bias de las n primeras de capas de neuronas a sus análogas en la nueva red neuronal. El resto de capas son inicializadas de forma aleatoria. El entrenamiento de esta nueva red se puede hacer mediante ajuste fino (*fine-tuning*) para adaptar las características reutilizadas a su nueva tarea, o mediante la congelación de estas capas pre-entrenadas para que su valor no cambie durante el entrenamiento

El uso de características congeladas es recomendable cuando el dataset es pequeño y hay muchos parámetros entrenables, de lo contrario la red puede tender al sobreajuste. Cuando el conjunto de datos de entrenamiento es muy grande y no hay muchos parámetros entrenables, un ajuste fino es la mejor opción ya que el riesgo de sobreajuste es menor.

3. Herramientas para la implementación del modelo y aprendizaje

La elección de las tecnologías principales a utilizar ha sido condicionada por el hecho de partir de un proyecto ya existente. Este proyecto fue implementado por primera por vez en 2013 en Python, haciendo uso de las librerías Theano[6] y Lasagne[7] para la implementación de redes neuronales. Cuando Google compró el grupo DeepMind, todo el proyecto fue traducido a Lua [3], usando las librerías Torch[8] y nn[9] para la implementación de las redes neuronales [12]. Indistintamente de las tecnologías usadas, ambos proyectos no tienen ninguna diferencia destacable respecto al funcionamiento de su aprendizaje. Ambos proyectos hacen uso del framework Arcade Learning Environment (ALE).

Finalmente, tras comparar ambas opciones se eligió el proyecto implementado en Lua, por ser el más reciente y porque a pesar de que Theano y Torch obtienen un rendimiento en GPU muy parecido [26], este último es ligeramente más eficaz en el entrenamiento de grandes redes convolucionales y redes de neuronas totalmente conectadas. No obstante, para futuros proyectos, también habría que tener en cuenta el framework TensorFlow, diseñado por Google, ya que DeepMind lo ha comenzado a usar como su framework de desarrollo[10]. Al no tener ninguna experiencia previa en el uso de estas tecnologías al comienzo del proyecto, el primer paso fue el aprendizaje de éstas y de las posibilidades que ofrecen en el desarrollo de algoritmos de aprendizaje automático. Para más detalles sobre a instalación del proyecto, sus dependencias y una descripción de las máquinas utilizadas, ver el Anexo B.

3.1. Lenguaje de implementación

Lua es un lenguaje de programación imperativo y estructurado que destaca por su ligereza, por lo que se adapta muy bien a los largos procesos de cálculo requeridos por las técnicas de aprendizaje automático. Es un lenguaje basado en C, lo que favorece su integración con ALE mediante el wrapper **AleWrap**, desarrollado por DeepMind.

Torch7 es un framework que ofrece a Lua un amplio abanico de algoritmos y utilidades matemáticas necesarias en aprendizaje automático. Torch también simplifica la programación orientada a objetos y su serialización, permitiendo así guardar el resultado de un entrenamiento de forma sencilla. Este framework es usado por empresas y grupos de investigación como el Facebook AI Researcher Group, IBM o Yandex.

[6] http://www.deeplearning.net/software/theano/
[7] https://lasagne.readthedocs.io/en/latest/
[8] http://torch.ch/
[9] https://nn.readthedocs.io/en/latest/
[10] http://googleresearch.blogspot.com.es/2016/04/deepmind-moves-to-tensorflow.html

3.1.1. Librerías utilizadas

nn: Utilizado para implementar las redes neuronales. Divide la red en módulos combinables para formar estructuras más complejas.

qtlua y qttorch: Proporcionan las herramientas necesarias para visualización de imágenes. Han sido utilizados para la visualización de la pantalla de juego y para mostrar el proceso y los resultados de los entrenamientos.

unsup: Proporciona herramientas para la implementación de aprendizaje no supervisado.

cutorch y cunn: Proporciona los tipos de datos, funciones y backend de los módulos de *nn* para trabajar en GPU con CUDA.

optim: Proporciona diferentes algoritmos para el entrenamiento de redes neuronales como el descenso de gradiente estocástico.

3.1.2. Atari 2600 y Arcade Learning Environment

Atari 2600 es una videoconsola de 1977 cuya popularidad y la de sus sucesoras se prolongó hasta comienzos de la década de los noventa. Sus juegos se controlan mediante un *joystick* y un botón para cada jugador, lo que permite una combinación de movimientos más acotada que otras consolas, y sus gráficos están formados por un máximo de 128 colores.

Debido a sus limitadas acciones y gráficos, ésta ha sido la principal consola utilizada para intentar imitar el comportamiento humano aplicado a videojuegos. Consecuencia de este movimientto, se ha desarrollado el *Arcade Learning Environment* (ALE) [27], un emulador de código abierto implementado en C++, orientado a la investigación y desarrollo de técnicas de aprendizaje automático. Este emulador permite a un agente interactuar con juegos del Atari 2600, mediante la ejecución de las acciones/movimientos que cada juego tiene disponibles. Como salida del framework, el agente recibe una imagen de la pantalla, la puntuación y el estado del juego. ALE, también ofrece herramientas para tratar el parpadeo de ciertos objetos de la pantalla, obtener el audio del juego, acceder a la RAM del emualdor o grabar partidas.

3.2. Herramientas auxiliares

También se ha usado **Java** para la obtención de estadísticas y gráficas a partir de los datos generados por los entrenamientos, y **C++** con **OpenCV** para la extracción de los frames que componen un video, con el fin de utilizarlos como entrada de un entrenamiento.

GitHub se utiliza como herramienta de control de versiones [11].

[11] https://github.com/asabater94/Atari-DeepReinforcementLearning

4. Replicación del proyecto

El desarrollo del proyecto ha comenzado con la replicación del trabajo realizado por Google DeepMind. Éste cuenta con dos versiones distintas cuya única diferencia destacable, a demás de las tecnologías usadas para su implementación (ver Sección 3), es la arquitectura de la red neuronal utilizada. En este proyecto se ha replicado la segunda versión del trabajo realizado por DeepMind [3], cuyo proceso de instalación está detallado en el Anexo B, y los experimentos realizados por el grupo de investigación. A continuación se describe el proceso de aprendizaje implementado por DeepMind. Posteriormente se define la arquitectura de la red neuronal utilizada, y se explican las modificaciones que se van a efectuar para su pre-entrenamiento.

4.1. Detalles del entrenamiento

El problema que aquí se aborda es el aprender a jugar a videojuegos de Atari 2600 mediante aprendizaje profundo por refuerzo. Para ello se usa una red neuronal convolucional, cuya función es la de calcular el valor Q* de cada acción a partir de los frames generados por el emulador, siendo la acción con mayor Q* la mejor acción a tomar por el agente. La estrategia seguida es evaluada utilizando como retroalimentación la recompensa obtenida por el emulador en cada momento.

El método de entrenamiento por refuerzo utilizado es el llamado Q-learning, por lo que en todo momento se mantiene un dataset con las últimas experiencias (estado, acción elegida, recompensa y estado siguiente) generadas por el emulador. La actualización de los pesos de la red neuronal se realiza sobre lotes de este dataset, siguiendo el algoritmo RMSProp [11], que utiliza la media de los gradientes recientes para normalizar los siguientes. El uso de este almacén, aumenta la eficiencia del entrenamiento y reduce la varianza de las actualizaciones y la divergencia del aprendizaje.

Los frames del juego, utilizados en el aprendizaje, que proporciona ALE están en formato RGB y tienen un tamaño de 210 x 160 x 3. Para reducir su dimensionalidad, se aplica un preprocesamiento que los convierte a una escala de grises, y los reescala a un tamaño de 84 x 84. Como a partir de un único frame del juego no es posible deducir la estrategia que siguen cada uno de sus componentes, la entrada de la red neuronal es la concatenación de los 4 últimos frames obtenidos del emulador. Entre cada par de frames, se desechan otros 4, con el fin de reducir muestras de entrenamiento similares y conseguir una mejor deducción del movimiento de cada uno de los objetos del juego en la pantalla. Al desecharlos, la acción elegida por el agente se propaga a los 4 frames siguientes. Cada uno de estos grupos de 4 frames forman un estado del sistema.

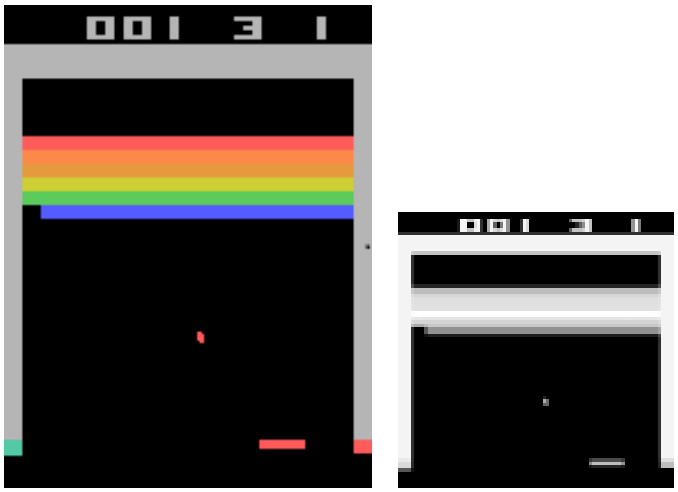


Figura 7: Pre-procesamiento de frames. En la izquierda la imagen original. En la derecha el resultado del pre-procesamiento de la imagen original.

Este tamaño fijo de 84 x 84 x 4, es la entrada de una red convolucional profunda formada por una serie de capas convolucionales (intercaladas con capas ReLU), seguidas por capas de neuronas totalmente conectadas, cuya capa de salida está compuesta por tantas neuronas como acciones tiene el juego. Cada salida genera el Q* predicho para una acción del juego.

Durante el entrenamiento, se selecciona cada acción siguiendo una estrategia ϵ-greedy, es decir aquella que ofrece un Q* más alto, con una probabilidad de $1 - \epsilon$ de seleccionar una acción aleatoria. Esta probabilidad decrece linealmente de 1 a 0.1, durante los 1.000.000 primeros frames, quedándose fija a 0.1 después de éstos. Esa aleatoriedad inducida permite una exploración más abierta de la estrategia a seguir por el agente.

Como el valor de las recompensas de cada juego es diferente, éstas son transformadas a 1, -1 ó 0, si tiene un valor positivo, negativo o nulo. Esto facilita el uso de los mismos parámetros de entrenamiento para cada juego, pero el hecho de que no reconozca recompensas de diferente magnitud afecta a los resultados del entrenamiento.

Para más detalles del entrenamiento, referimos al lector a los papers escritos por DeepMind en arXiv [12] y Nature [3].

4.2. Arquitectura de la red neuronal utilizada

La segunda versión del proyecto de DeepMind [3] ha sido replicada conservando el valor de todos los parámetros de entrenamiento definidos por el grupo

de investigación. Sin embargo, la arquitectura de la red neuronal se ha modificado.

La red neuronal de la primera versión del proyecto estaba formada por dos capas convolucionales con 16 filtros de tamaño 8x8 y 32 filtros de tamaño 4x4 respectivamente, ambas seguidas por el rectificador ReLU. Y seguidas a éstas, dos capas de neuronas totalmente conectadas, una de 256 neuronas y otra con tantas neuronas como acciones tiene el juego a entrenar.

La segunda versión del proyecto tiene una arquitectura formada por tres capas convolucionales de 32 filtros de tamaño 8x8, 64 filtros de tamaño 4x4, y 64 filtros de tamaño 3x3, todas ellas seguidas por el rectificador ReLU. Seguida a la parte convolucional hay dos capas de neuronas totalmente conectadas, una con 512 neuronas y otra con tantas neuronas como acciones tiene el juego a entrenar.

Debido a los buenos resultados demostrados, se intentó replicar el segundo modelo, pero éste necesitaba una cantidad de memoria que las máquinas de entrenamiento no podían soportar, por lo que finalmente se creó un modelo de un tamaño intermedio. Tras un estudio de la eficiencia de diferentes arquitecturas de redes convolucionales profundas [14], se decidió invertir la memoria disponible en aumentar el tamaño de las capas convolucionales, y no crear más de éstas. Finalmente el modelo utilizado está formado por dos capas convolucionales de 32 filtros de tamaño 8x8 con stride 4 y 64 filtros de tamaño 4x4 con stride 2, respectivamente, iguales a las dos primeras capas del segundo modelo descrito. Contiguas a éstas, dos capas de neuronas totalmente conectadas, una de 512 neuronas y otra con tantas neuronas como acciones tiene el juego a entrenar. No se han usado capas de agrupación, ya que ésto transmitiría una cierta invarianza a la traslación de las características entrenadas perjudicando la toma de decisiones en el sentido de que la ubicación de un objeto en la escena es muy importante.

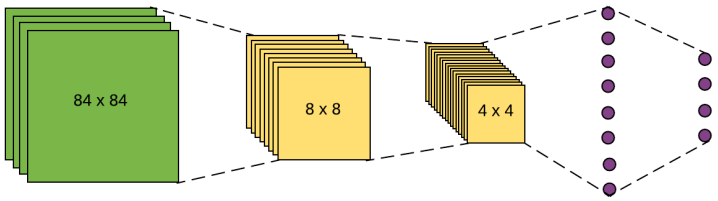
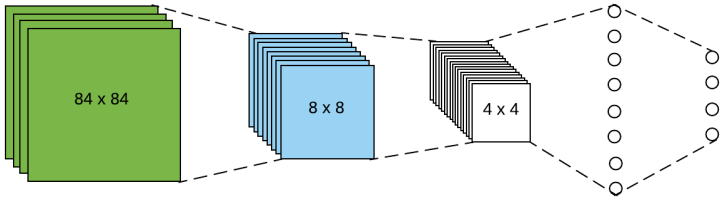


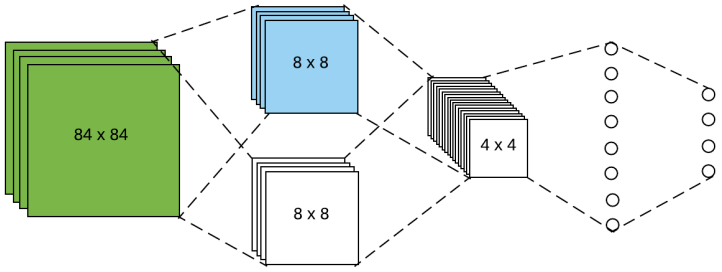
Figura 8: Esquema de la red neuronal utilizada. En verde, la entrada de la red. En amarillo, los filtros convolucionales. En morado, las neuronas totalmente conectadas.

4.3. Adaptación de la red neuronal a los pesos pre-entrenados

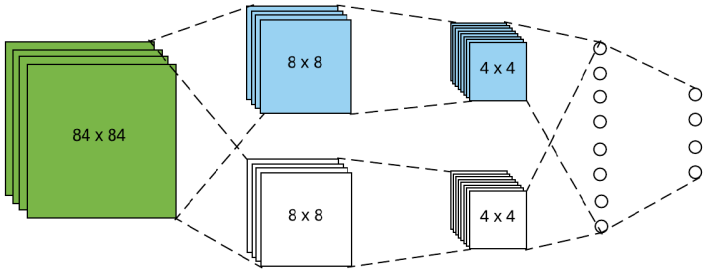
Los pesos pre-entrenados a utilizar están almacenados en las capas de una red neuronal, con las mismas características que sus análogas en el modelo principal. Para cargarlos, se clonan las capas entrenadas a la nueva red (pesos y bias) modificando su arquitectura en función del tamaño y número de filtros pre-entrenados (ver Figura 9).



(a) Pre-entrenamiento de una capa convolucional



(b) Pre-entrenamiento de media capa convolucional



(c) Pre-entrenamiento de dos medias capas convolucionales

Figura 9: Esquema de pre-entrenamiento de capas de neuronas. En verde, la entrada de la red. En azul, los filtros de neuronas pre-entrenadas

Cuando una de las capas convolucionales reutilizadas tiene un número de filtros inferior al deseado, se crea una nueva capa paralela a la pre-entrenada con el número de filtros necesarios para completar la especificación del modelo descrito en la sección anterior. Estos filtros que completan el modelo son inicializados de

forma aleatoria. La salida de dos capas convolucionales paralelas generan dos vectores multidimensionales de neuronas. Si la siguiente capa de neuronas tiene otra capa en paralelo, cada una de estas capas toma como entrada el conjunto de neuronas que le preceden en su mismo nivel (ver Figura 9c). Si no tiene ninguna capa en paralelo, su entrada es la formada por la concatenación de los vectores generados por las dos capas paralelas anteriores (ver Figura 9b).

La red neuronal está implementada en Torch mediante módulos, de forma que cada uno de los componentes de la red (capas de neuronas, funciones de activación, etc) corresponde a un módulo. La actualización de los pesos y bias de los módulos entrenables se realiza mediante las funciones *accGradParameters* y *updateParemters*, específicas para cada uno de ellos. Para congelar los pesos de una determinada capa, estas funciones son eliminadas del módulo que las representa, por lo que durante el proceso de aprendizaje, el valor inicial de estos módulos congelados se mantiene constante.

5. Implementación del aprendizaje no supervisado

Debido a las restricciones de tiempo y capacidad de computación, únicamente se ha implementado el aprendizaje de características mediante K-means y autocodificadores, cuyos de detalles se describen a continuación. Cada aprendizaje no supervisado únicamente toma como datos de entrada los frames del juego para el que se va a entrenar.

5.1. Datos de entrenamiento

Los datos de entrenamiento del aprendizaje no supervisado han sido obtenidos de partidas grabadas de juegos de Atari 2600, subidas por la comunidad a YouTube. De estos vídeos se han extraído sus frames, que han sido procesados para adaptarse a los requisitos de la red neuronal, explicados en la Sección 4.

La herramienta utilizada para la extracción de frames ha sido OpenCV, cuya única función es la de eliminar los márgenes de los frames y guardarlos en formato PNG. En lugar de utilizar OpenCV en el tratamiento de las imágenes, para asegurar que el resultado del pre-procesamiento de estas imágenes es el mismo que el realizado a los frames obtenidos del emulador durante el aprendizaje, el código en Lua que lo lleva a cabo durante el entrenamiento ha sido adaptado para esta nueva tarea. Este código transforma las imágenes a escala de grises extrayendo su canal Y (luminancia) y las re-escala a un tamaño de 84x84. Debido a la diferente frecuencia de generación de frames entre el emulador y los videos obtenidos, en lugar de utilizar uno de cada 5 frames, este valor se ha ajustado de forma empírica para cada juego.

Como resultado de este pre-procesamiento, los datos generados son guardados en una matriz multidimensional de tamaño nx84x84x4, siendo n el número de grupos de 4 frames generados. Cada uno de estos grupos constituye un dato de entrenamiento para los algoritmos de aprendizaje no supervisado.

Los datos de entrenamiento no han sido normalizados ni sometidos a whitening [28] ya que al no ser imágenes naturales, no tienen ningún cambio de iluminación ni color.

5.2. K-means

El algoritmo de K-means utilizado es el disponible en el paquete unsup de Torch, ligeramente modificado[12] para adaptarse a las especificaciones de Adam Coates en [14]. Esta adaptación facilita la eliminación del clústers vacíos.

[12] https://github.com/jhjin/kmeans-learning-torch

La entrada de K-means está formada por un conjunto de trozos (*patches*) de datos de entrenamiento obtenidos de forma aleatoria sobre éstos (por defecto 50.000). Estos datos tienen el mismo tamaño que los filtros de la capa convolucional a la que van destinados. Los centroides, que también tienen el mismo tamaño que los filtros, son inicializados de forma aleatoria siguiendo una distribución normal entorno a 0. El valor k del algoritmo (número de centroides), corresponde al número de filtros a entrenar.

Tras la primera ejecución de K-means, se añade una nueva capa convolucional con los filtros aprendidos a la red neuronal a pre-entrenar, inicialmente vacía. Para el aprendizaje de los filtros de la siguiente capa, los datos de entrenamiento iniciales son procesados por la red pre-entrenada, y sobre el resultado obtenido se extraen los nuevos trozos con los que se entrena el K-means de la segunda capa convolucional, que también será añadida a la red pre-entrenada. Este proceso se repite hasta tener entrenados los filtros de todas las capas deseadas. Como los bias no son entrenados, se inicializan de forma aleatoria en cada una de las capas.

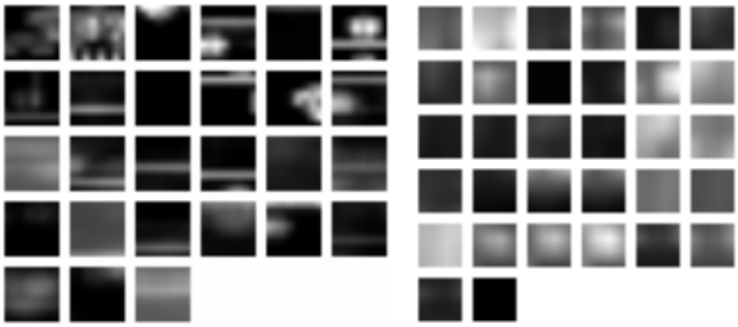


Figura 10: Muestra de filtros aprendidos por K-means en la primera y segunda capa convolucional.

La ejecución de K-means puede dar lugar al aprendizaje de clústers vacíos, que son descartados ya que no van a aportar ninguna información de interés al entrenamiento principal. Para reducir el número de clústers vacíos, se realizan varias repeticiones de la ejecución del algoritmo al que se le pide el aprendizaje de más filtros. No obstante, se ha observado que la búsqueda de filtros no vacíos está condicionada por los datos de entrenamiento y el tamaño de los filtros a aprender. Las capas más profundas de la red neuronal utilizada, donde los filtros son más pequeños y se entrenan más centroides, son más propensas a generar clústers vacíos.

Como resultado de este aprendizaje, se han aprendido un conjunto de filtros de forma jerárquica siendo los pertenecientes a capas más superficiales los destinados a reconocer características más generales de la imagen. La red entrenada es almacenada en un fichero para servir de pre-entrenamiento a un nuevo modelo.

El hecho de no tener bias pre-entrenados, no supone un problema en la realización de un ajuste fino, ya que estos serán modificados para adaptarse a la tarea que les es asignada. Sin embargo en la congelación de estas capas pre-entrenadas, la existencia de bias aleatorios puede generar ruido en el aprendizaje. Para solucionarlo, se hace uso de uno de los módulos que ofrece Torch. Este componente se llama *Add* y básicamente es una capa de la red que genera como salida la suma de un escalar entrenable a cada uno de sus datos de entrada, es decir, hace la función de un bias. Esta nueva capa es añadida después de una capa de neuronas pre-entrenada con K-means.

5.3. Autocodificadores

La arquitectura del codificador a entrenar está formada por una o más capas convolucionales con los mismos parámetros que sus análogas en el modelo real de entrenamiento. Por cada capa del codificador hay una capa totalmente convolucional del mismo tamaño en el decodificador, de esta forma la primera capa del codificador es la complementaria de la última del decodificador. Esta relación se propaga de los extremos hacia el interior del autocodificador (ver la Figura 6 en la Sección 2.3.1).

Esta red neuronal es entrenada mediante un descenso de gradiente estocástico que busca igualar la salida de la red a su entrada. Para ello se hace uso del paquete *optim* de Torch que a partir de una red neuronal, un dataset y los parámetros de entrenamiento (tasa de aprendizaje, reducción de la tasa de aprendizaje y epochs) trata de minimizar el valor obtenido por la función de estimación de error definida. Este algoritmo ha sido modificado para tener acceso al error estimado en cada momento del aprendizaje y mostrar en una gráfica los resultados obtenidos de éste.

El dataset está compuesto por el conjunto de datos de entrenamiento definido en la Sección 5.1 y el resultado esperado de cada uno de ellos como salida de la red neuronal. Ambos conjuntos de datos son los mismos, ya que se busca una reconstrucción idéntica a la entrada de la red. La función de aproximación utilizada durante el aprendizaje es el error medio cuadrático (4), que estima el error de reconstrucción calculando la diferencia *píxel* a *píxel* entre la entrada y la salida de la red.

Para mejorar el rendimiento del algoritmo, se ha probado el aprendizaje utilizando una función de activación al comienzo del decodificador. Tras un estudio de las diferentes funciones de activación utilizadas por la comunidad en el entrenamiento de autocodificadores se ha probado la eficiencia del uso de la función logística [21], la tangente hiperbólica [22] y la función *Diag* [29] (que multiplica el resultado de la tangente hiperbólica por una matriz diagonal de valores entrenables). Tras analizar los resultados obtenidos del entrenamiento de autocodificadores de una y dos capas convolucionales con alguna de las funciones de

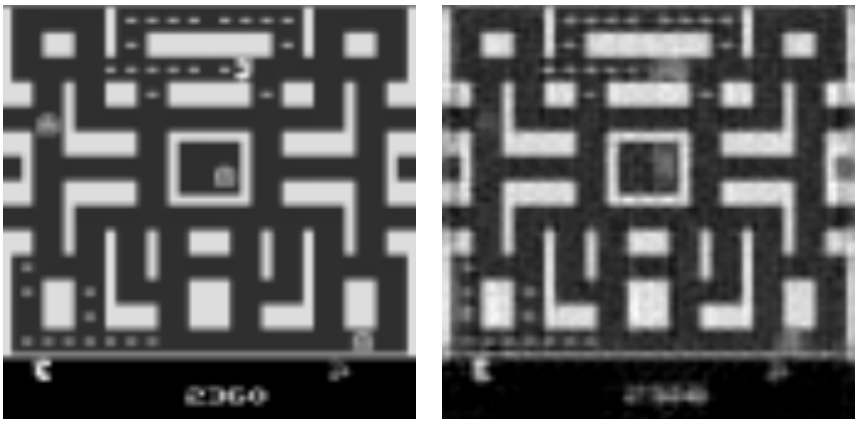


Figura 11: Ejemplo de reconstrucción de la imagen original a partir de los filtros aprendidos por un autocodificador.

activación (o ninguna), con el error generado en cada momento del aprendizaje se ha visto que los mejores resultados son los obtenidos por aquellos modelos que no usan ninguna función de activación. En la Figura 12 se puede ver la comparación del entrenamiento de un modelo de dos capas convolucionales con las diferentes funciones de activación mencionadas. También se ha probado la calidad del entrenamientos con datos normalizados, y su resultado da lugar a un error 200 veces peor que el generado por el entrenamiento de la misma arquitectura con los datos sin normalizar. Para más detalles de los resultados de las pruebas realizadas, ver el Anexo C.

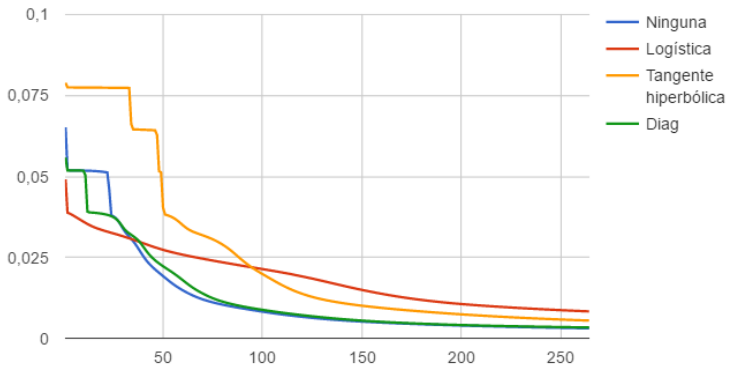


Figura 12: Aprendizaje con diferentes funciones de activación.

Finalmente, la arquitectura del autocodificador utilizado para el pre-entrenamiento de la red neuronal principal está formado por el número y tamaño de capas convolucionales exigido por el modelo del entrenamiento principal (con capas ReLU

intercaladas), sin ninguna función de activación de las anteriormente mencionadas. A este modelo también se le ha añadido una capa de *dropout espacial* al comienzo del decodificador, cuyo objetivo es favorecer el aprendizaje de características más dispersas y evitar el sobreajuste de la red neuronal.

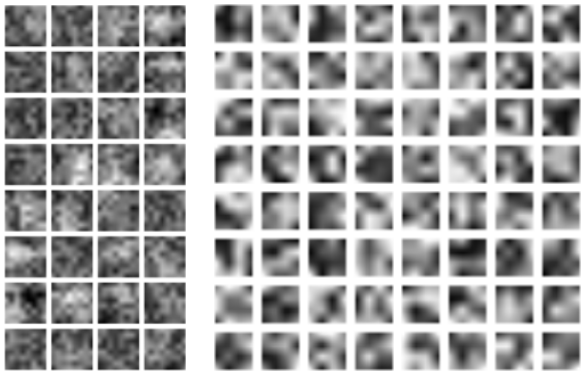


Figura 13: Muestra de filtros aprendidos por un autocodificador en la primera y segunda capa convolucional.

6. Resultados

Las pruebas realizadas se basan en los resultados obtenidos de la replicación del proyecto con la nueva arquitectura de la red neuronal, y la comparación de su rendimiento con el obtenido mediante distintas técnicas de pre-entrenamiento de la red neuronal. Los juegos sobre los que se realizan estas pruebas son el *Space Invaders*, *Demon Attack* y *Phoenix*, que comparten la misma temática y tienen unos gráficos similares, lo cual nos interesa para probar la trasferencia de conocimiento (ver Figura 14).

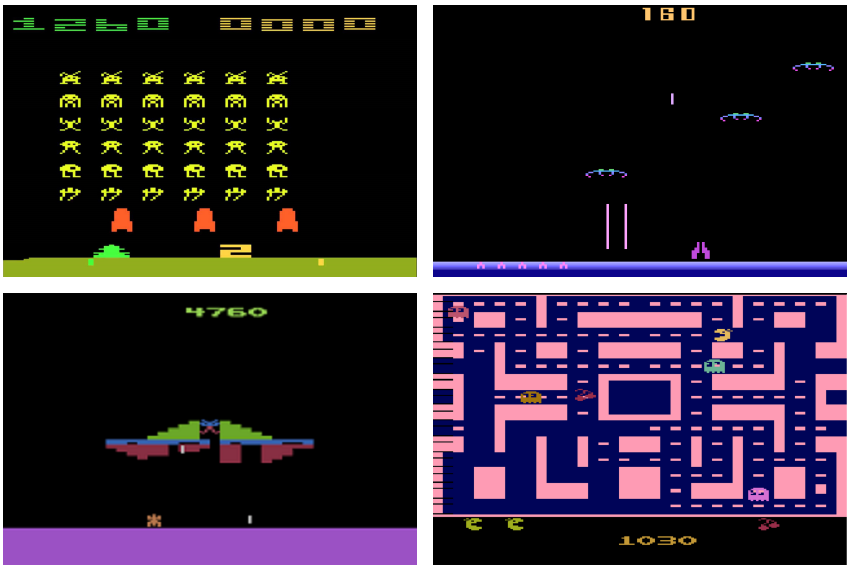


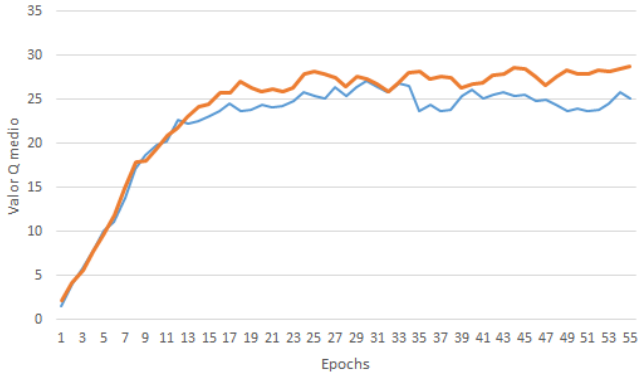
Figura 14: Juegos utilizados. Arriba, *Space Invaders* y *Demon Attack*. Abajo, *Phoenix* y *Ms Pacman*.

De acuerdo con los resultados obtenidos por el grupo DeepMind [3], el aprendizaje de estos juegos muestra resultados superiores a los conseguidos por un humano profesional, salvo el *Phoenix* del que no se tiene información. También se han realizado pruebas con *Ms Pacman*, cuyos resultados obtenidos por el grupo DeepMind reflejan un aprendizaje deficiente en comparación con los obtenidos por un humano. Parte de la complejidad del entrenamiento radica en el número de acciones a entrenar para cada juego, reflejadas en el Cuadro 1.

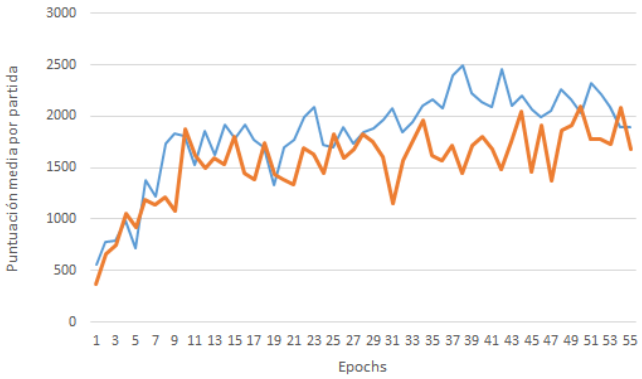
	Número de acciones
Space Invaders	6
Demon Attack	6
Phoenix	8
Ms Pacman	9

Cuadro 1: Número de acciones de los juegos entrenados

Para comprobar la calidad de un entrenamiento se toman dos medidas: la media de los últimos valores Q obtenidos en cada momento del aprendizaje, cuyo valor se quiere maximizar, y la puntuación media de juego obtenida durante la evaluación del aprendizaje. Esta evaluación se realiza de forma periódica durante el proceso de aprendizaje, y en ésta, el agente selecciona para cada estado la acción cuyo valor Q* obtenido de la red es el máximo de entre todas las acciones del juego. La puntuación final es la media de las puntuaciones de todas las partidas en las que ha jugado durante 125.000 frames. Con estas medidas se puede hacer una comparación de la velocidad de convergencia y resultados obtenidos entre varios entrenamientos.



(a) Media del valor Q predicho



(b) Puntuación media obtenida por partida

Figura 15: Entrenamiento de Ms Pacman con 16 filtros pre-entrenados y congelados en la primera capa convolucional. En azul, el entrenamiento original. En naranja, pre-entrenado con autocodificadores.

6.1. Resultados de la replicación del proyecto

La valoración de los resultados del nuevo modelo de red neuronal definida para el proyecto, se ha realizado mediante un entrenamiento para cada uno de los juegos mencionados anteriormente y la comparación de sus resultados con los obtenidos por DeepMind en [3]. El método de evaluación utilizado es el mismo que el utilizado por el grupo de investigación, y consiste en la ponderación de la puntuación obtenida en el proceso de evaluación de una DQN con la obtenida en partidas de un humano profesional y en un agente que toma acciones aleatorias, de acuerdo con la fórmula (5). Los resultados obtenidos se pueden ver en el Cuadro 2, donde se recoge la valoración respecto a la puntuación final obtenida en el entrenamiento, y la máxima alcanzada en éste.

$$\frac{100 * (puntuacion\_DQN - puntuacion\_agente\_aleatorio)}{(puntuacion\_humano - puntuacion\_agente\_aleatorio)} \tag{5}$$

	Valoración obtenida	Máxima valoración obtenida	Valoración obtenida por DeepMind
Space invaders	89 %	104 %	121 %
Demon Attack	210 %	267 %	294 %
Ms Pacman	11 %	14 %	13 %

Cuadro 2: Valoración obtenida de la replicación del proyecto.

A la vista de los resultados obtenidos, se puede observar que el nuevo diseño de la red, a pesar de contar con una capa convolucional menos, obtiene resultados casi tan buenos como los obtenidos por el grupo DeepMind, llegando a superar a un humano (salvo en el caso de *Ms Pacman*). El juego *Phoenix* no ha sido evaluado por el desconocimiento de la puntuación obtenida por un humano y un agente aleatorio, no obstante se ha comprobado de forma empírica que los resultados obtenidos podrían superar a los obtenidos por un humano.

6.2. Pruebas de la red neuronal pre-entrenada de forma no supervisada

Para la evaluación del uso de características entrenadas de forma no supervisada, se ha entrenado el modelo con variaciones en el número de filtros pre-entrenados, con y sin congelar. Con la congelación de los pesos de una determinada capa de neuronas se evita su aprendizaje, lo que aumentaría la velocidad de aprendizaje del entrenamiento.

Ms Pacman

Este juego ha sido probado en un modelo con todas las capas convolucionales pre-entrenadas, un modelo con una capa convolucional pre-entrenada y congelada, y un modelo con la mitad de la primera capa convolucional (16 filtros) congelada. El pre-entrenamiento de cada uno de estos tipos de aprendizaje ha sido llevado a cabo con características obtenidas de autocodificadores y K-means.

Phoenix

Para este juego se ha probado el rendimiento de un modelo con las dos capas convolucionales pre-entrenadas y una red con la mitad de la primera capa pre-entrenada y congelada, ambas inicializaciones realizadas con autocodificadores.

6.3. Pruebas de la red neuronal pre-entrenada de forma supervisada

Las pruebas de transferencia de conocimiento se han realizado sobre los juegos *Space Invaders*, *Phoenix* y *Demon Attack*. Para ello, el aprendizaje de una nueva red neuronal ha recibido la información de otra red entrenada con un juego que no ha recibido ningún tipo de pre-entrenamiento.

Space Invaders

Las pruebas realizadas sobre este juego se basan en un aprendizaje inicializado con la información de todas las capas convolucionales y totalmente conectadas de un modelo entrenado para jugar al *Demon Attack*, y en un entrenamiento con la primera capa convolucional de la red congelada y pre-entrenada a partir de este mismo modelo.

Demon Attack

Este juego ha sido entrenado partiendo del conocimiento de todas las capas convolucionales y totalmente conectadas de una red neuronal entrenada para jugar al *Space Invaders*. Otro entrenamiento ha sido realizado inicializando las capas convolucionales a partir de un modelo entrenado para jugar al *Phoenix*.

6.4. Análisis de resultados

De los resultados obtenidos, presentes en el Anexo D, se puede deducir que la puntuación de juego obtenida con el uso de pesos pre-entrenados de forma no supervisada no ha producido la mejora esperada. Los mejores resultados se han obtenido con el uso del pre-entrenamiento de autocodificadores llegando incluso a superar el resultado original en su uso para pre-entrenar toda la parte convolucional de la red. No obstante, en la visualización de la media de los valores Q obtenidos, si que se puede observar su incremento en el caso de Ms Pacman y una convergencia más rápida en el caso de Phoenix.

El uso de K-means, ha perjudicado el entrenamiento, obteniendo unos resultados muy por debajo de los originales. Esto puede ser debido a su entrenamiento con imágenes no naturales, en el que el aprendizaje de gradientes de color u otras formas es más complicado, como se puede observar en la Figura 10.

En los entrenamientos con trasferencia de conocimiento, en todos los casos se ha conseguido una mejora de los valores Q obtenidos, favoreciendo la convergencia del aprendizaje. No obstante, las puntuaciones de juego obtenidas en el Space Invaders, a pesar de conseguir una convergencia mucho más rápida, su valor final es menor que los resultados originales. Sin embargo, en el caso de Phoenix, se obtiene un aprendizaje más progresivo, llegando a igualar los resultados originales.

En lo relativo a la congelación de filtros convolucionales, los resultados muestran una mejora en los valores Q obtenidos, consiguiendo un aumento de su valor, o una convergencia más rápida. La puntuación de juego obtenida para cada entrenamiento, muestra que los mejores resultados se obtienen en la congelación de media capa convolucional. La congelación de una capa entera, restringe mucho la capacidad de elección de los filtros adecuados por parte del agente, reflejando este problema en los resultados obtenidos.

De los resultados obtenidos, también se puede deducir que en el proceso de aprendizaje de las capas convolucionales, además de aprender las propiedades relativas a la visualización de los frames, también se aprenden otras características que no se pueden aprender previamente, cuya implicación es específica para la tarea asignada a la red neuronal. También se puede observar como la ejecución de las mismas pruebas dan resultados diferentes para distintos juegos, lo que refleja las distintas necesidades de aprendizaje que tiene cada uno. Finalmente, y de acuerdo con las conclusiones obtenidas por [14], se puede observar, como la mayor restricción está en la arquitectura de la red neuronal elegida y la complejidad del problema a resolver.

7. Conclusiones

Basándose en el entrenamiento por refuerzo de una agente cuya tarea es aprender a jugar a videojuegos, he reforzado la base teórica relativa a las redes neuronales y el aprendizaje automático. Tras un análisis exhaustivo del proceso de aprendizaje de este agente, se ha estudiado el uso de técnicas que ayuden a mejorar el rendimiento y los resultados obtenidos de su aprendizaje. En este proceso he adquirido nuevos conocimientos de técnicas más elaboradas referentes a las redes convolucionales profundas, sobre las que he aprendido a identificar sus límites y las diferentes técnicas a emplear para aprovechar todo su potencial.

De entre estas técnicas, se ha hecho especial hincapié en el estudio del aprendizaje por refuerzo y en los diferentes tipos de entrenamiento supervisado y no supervisado para el aprendizaje de características, reutilizables en la inicialización de nuevos modelos.

Fruto de este estudio, he implementado el aprendizaje de características mediante K-means y autocodificadores; y he utilizado la transferencia de conocimiento entre redes neuronales entrenadas para diferentes juegos. Para el aprovechamiento de esta información, he definido nuevas arquitecturas, cuya eficacia se ha probado mediante el diseño y ejecución de una batería de pruebas, acorde con los recursos disponibles, que incluye el entrenamiento de diferentes juegos utilizando diferentes tipos de pre-entrenamiento de la red neuronal. Del análisis de los resultados obtenidos (ver Anexo D) se van a proponer nuevas líneas de trabajo que continúen la búsqueda de una mejora del rendimiento en el entrenamiento por refuerzo.

De la parte del desarrollo, he adquirido los conocimientos de Lua y su framework Torch que permiten aprovechar su potencial como herramientas de desarrollo referentes en el marco de la programación de algoritmos de aprendizaje automático. También aprendido los requisitos que éstos necesitan a nivel de hardware y software para su integración en diferentes máquinas.

Como resultado de este proyecto también he mejorado las competencias profesionales relacionadas con el autoaprendizaje y la gestión de proyectos, tomando consciencia de los retos, riesgos y limitaciones a abordar durante el desarrollo de éstos.

7.1. Trabajo futuro

Una continuación del proyecto empezaría extendiendo la batería de pruebas diseñada a nuevos juegos e introduciendo nuevas técnicas de aprendizaje de características como las redes de creencia profunda.

Para reforzar el uso de los filtros pre-entrenados en un nuevo modelo, propongo la modificación del proceso de entrenamiento para que estos filtros sean congelados temporalmente mientras se 'ajustan' los pesos no pre-entrenados, y tras una descongelación, la red entera se entrenaría adaptando los pesos necesarios para optimizar la tarea que le es encomendada.

Otra forma distinta de abordar este problema sería hacer uso de otra red neuronal que aprenda a segmentar los frames obtenidos por el emulador, de forma análoga al trabajo realizado en [8, 30]. Esta nueva información segmentada con los objetos útiles de cada frame, sería la nueva entrada de la red entrenada con aprendizaje por refuerzo. De esta forma, al tener la información útil acotada, se podría reducir el tamaño de la red neuronal y sus datos de entrada, lo que reduciría el tiempo de entrenamiento y el sobreajuste que éste puede generar.

7.2. Cronograma

La evolución de este proyecto se ha visto condicionada y limitada por diferentes factores. En primer lugar la compatibilidad de su desarrollo con el periodo lectivo ha impedido una implicación a tiempo completo durante la mayor parte del mismo. El proceso de instalación de las herramientas y tecnologías a utilizar ha sido mayor que el esperado debido a la falta de compatibilidad con los recursos existentes, la instalación y configuración de sus dependencias, corrección de errores y el uso de diferentes máquinas.

La ejecución de las pruebas se ha visto retrasada por la falta de capacidad de computación. Hasta la última etapa del proyecto, únicamente se contaba con un ordenador para su desarrollo. Posteriormente, la incorporación de más material ha permitido la paralelización de las pruebas realizadas, algunas de ellas condicionadas por el uso compartido de las máquinas.

A continuación se expone la planificación seguida en el desarrollo del proyecto y las horas invertidas para cada tarea.

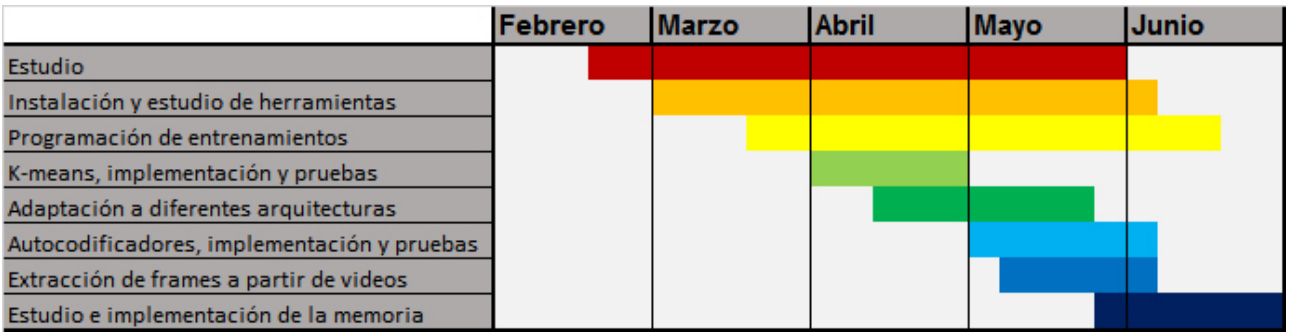


Figura 16: Planificación seguida durante el proyecto.

Tarea	Horas invertidas
Estudio	98
Instalación y estudio de herramientas	76
Programación de entrenamientos	22
Implementación de K-means y carga de pesos	27
Adaptación a diferentes arquitecturas	20
Autoencoders pruebas e implementación	36
Extracción de frames a partir de video	14
Estudio e implementación de la memoria	105

Cuadro 3: Horas invertidas en cada tarea.

Referencias

[1] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[2] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *J. Mach. Learn. Res.*, 10:1–40, June 2009.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[4] Anil K. Jain, Jianchang Mao, and K. Mohiuddin. Artificial neural networks: A tutorial. *IEEE Computer*, 29:31–44, 1996.

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, December 1989.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[7] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 648–656, 2015.

[8] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CVPR (to appear)*, November 2015.

[9] Charles M. Grinstead and J. Laurie Snell. Introduction to probability. chapter 11. AMS, 2003.

[10] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[11] Kevin Swersky Geoffrey Hinton, Nitish Srivastava. Neural networks for machine learning. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`, 2014. Accessed: 2016-06-07.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. *ArXiv e-prints*, December 2013.

[13] Jim Y. F. Yam and Tommy W. S. Chow. A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing*, 30(1-4):219–232, 2000.

[14] A. Coates, H. Lee, and A.Y. Ng. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*.

[15] Chuan Yu Foo Yifan Mai Caroline Suen Andrew Ng, Jiquan Ngiam. Unsupervised feature learning and deep learnign. `http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial`, 2016. Accessed: 2016-06-14.

[16] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability - Vol. 1*, pages 281–297. University of California Press, Berkeley, CA, USA, 1967.

[17] Jeff Schneider. Cross validation. `https://www.cs.cmu.edu/~schneide/tut5/node42.html`, 1997. Accessed: 2016-06-13.

[18] Christos Boutsidis, Petros Drineas, and Michael W Mahoney. Unsupervised feature selection for the k-means clustering problem. In *Advances in Neural Information Processing Systems 22*, pages 153–161. Curran Associates, Inc., 2009.

[19] Adam Coates and Andrew Y. Ng. Learning feature representations with k-means. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 561–580. Springer, 2012.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Internal Representations by Error Propagation, pages 673–695. MIT Press, Cambridge, MA, USA, 1988.

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

[22] Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I*, ICANN'11, pages 52–59, Berlin, Heidelberg, 2011. Springer-Verlag.

[23] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representation*, 2016.

[24] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? pages 3320–3328, 2014.

[25] Steven Gutstein, Olac Fuentes, and Eric Freudenthal. Knowledge transfer in deep convolutional neural nets. *International Journal on Artificial Intelligence Tools*, 17(3):555–567, 2008.

[26] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. Comparative study of caffe, neon, theano, and torch for deep learning. *CoRR*, abs/1511.06435, 2015.

[27] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 06 2013.

[28] Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, and Koray Kavukcuoglu. Natural neural networks. 2015.

[29] Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 399–406, 2010.

[30] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. *arXiv preprint arXiv:1505.04366*, 2015.

[31] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. Also published as a book. Now Publishers, 2009.

[32] Michael Nielsen. How the backpropagation algorithm works. `http://neuralnetworksanddeeplearning.com/chap2.html`, 2016. Accessed: 2016-06-14.

[33] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[34] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Commun. ACM*, 54(10):95–103, October 2011.

Anexos

A. Máquinas restringidas de Boltzmann y redes de creencia profundas

Debido a los límites de tiempo y capacidad de computación, no se ha podido implementar las redes de creencia profunda. No obstante, a pesar de no tener una implicación directa en el proyecto, esta técnica ha sido estudiada y se considera que es un candidato para el aprendizaje de características tan bueno como los autocodificadores y K-means. A continuación se explica la teoría subyacente a las máquinas restringidas de Boltzmann y redes de creencia profunda, y su entrenamiento.

Una máquina de Boltzmann restringida (*Restricted Boltzmann Machine*, RBM) [31] se puede considerar una red neuronal formada por una capa de neuronas visibles (entrada de la red) conectada a una capa de neuronas ocultas (salida de la red). A diferencia de las máquinas de Boltzmann no restringidas, las neuronas de una misma capa en una RBM no están relacionadas entre sí, lo que permite el uso de algoritmos de aprendizaje más eficientes. El objetivo de esta red es obtener una representación de su entrada, generalmente reduciendo su dimensionalidad. Los pesos que conectan la capa visible con la capa oculta de la red, mantienen una cierta relación estadística (para más detalles de la teoría matemática subyacente, referimos al lector al siguiente informe técnico [31]).

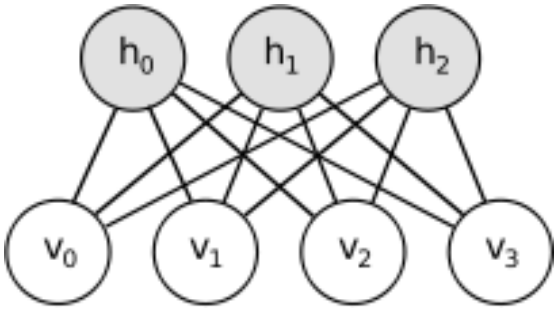


Figura 17: Estructura de una RBM. Los nodos v_i corresponden a las neuronas visibles, y los nodos h_i a las neuronas ocultas[13]

El entrenamiento de este modelo trata de reconstruir la entrada de la red mediante retropropagación (*backpropagation*) [32]. Para ello, con una propagación hacía delante se obtiene la activación de las neuronas y con una propagación hacia atrás se obtiene la reconstrucción generada por la representación creada

[13] `http://deeplearning.net/tutorial/rbm.html`

en la primera propagación. Esta reconstrucción es comparada con la entrada original de la red calculando un error de reconstrucción utilizado para actualizar los pesos del modelo. El objetivo de este aprendizaje es el de minimizar este error de reconstrucción.

Tras el entrenamiento, los pesos aprendidos corresponden a unas características capaces de detectar patrones en datos de entrenamiento no etiquetados. Estas características pueden ser utilizados como pesos pre-entrenados de una nueva red neuronal de forma análoga al aprendizaje mediante autocodificadores.

Para poder aprender características aplicables a múltiples capas neuronales de forma jerárquica, es necesario entrenar un nuevo modelo denominado red de creencia profunda (*Deep Belief Network*, DBN) [31] formado por la concatenación de múltiples RBM's.

Este modelo se entrena capa por capa [33], de forma que la capa oculta de una RBM es la capa visible de la siguiente. Cuando una RBM es entrenada, sus pesos y bias son congelados y el procesado de los datos de entrenamiento a través de esta RBM se toma como datos de entrenamiento para la siguiente. El esquema de este entrenamiento se puede ver en la Figura 18. Este proceso se repite hasta que todas las capas de la red neuronal se han entrenado. Las DBN son capaces de obtener resultados muy precisos a partir de pocas muestras de entrenamiento, consiguiendo terminar su aprendizaje en un corto periodo de tiempo.

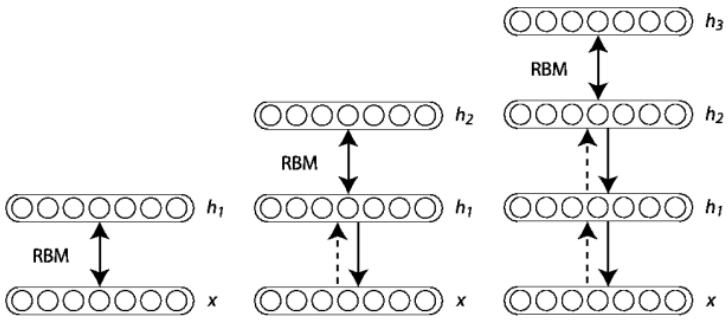


Figura 18: Entrenamiento de una red de creencia profunda[14].

Estas características aprendidas, guardan una estructura jerárquica, que puede ser usada como pre-entrenamiento de redes neuronales multicapa.

[14] `http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepBeliefNetworks`

Para explotar la correlación espacial que pueden tener los datos de entrenamiento, las RBM son sustituidas por máquinas de Boltzmann restringudas convolucionales (*Convolutional Restricted Boltzmann Machines*, CBRM), que de forma análoga a las capas de neuronas convolucionales, generan una salida por la convolución de diferentes filtros sobre los datos de entrada de la red. La concatenación de múltiples CRBM formaría una red de creencia profunda convolucional (*Convolutional Deep Belief Network*, CDBN) [34], cuyo entrenamiento se puede utilizar para inicializar un conjunto de capas convolucionales.

B. Instalación del entorno de entrenamiento

Google DeepMind proporciona un script para la instalación del proyecto con sus dependencias. La máquina sobre la que se ha instalado, en un inicio era un Ubuntu 15.10, pero como esta versión carecía del soporte necesario para la instalación de algunas dependencias, se actualizó a Ubuntu 16.04. Este ordenador cuenta con una tarjeta gráfica Nvidia GeForce 550 Ti. Para que los aprendizajes pudiesen hacer uso de GPU, ha sido necesario instalar el nvidia-cuda-toolkit y las librerías cutorch y cunn de Torch, que han requerido de una versión de gcc y g++ inferior a la 5.0 para su compilación. Los detalles de las tarjetas gráficas implicadas en este proyecto se pueden ver en e Cuadro 4.

Una vez instaladas las dependencias que permiten el uso de la GPU con CUDA, ha sido también necesario modificar el código original ya que hacía uso de funciones de cutorch y cunn obsoletas, lo que impedía su correcto funcionamiento. Concretamente, se ha modificado la creación de las capas convolucionales de la red neuronal, por su versión más reciente, compatible tanto con CPU como con GPU.

Adicionalmente a las dependencias que proporcionaba el código original, se ha instalado qtlua y qttorch para poder habilitar la opción de la visualización de la pantalla de juego. Para ello también ha sido necesario modificar el código original de AleWrap, que inicialmente bloqueaba esta posibilidad.

Los juegos de Atari 2600 han sido descargados de fuentes de terceros y su nombre también ha tenido que ser modificado para adaptarse a los requerimientos del emulador (ALE), cuya especificación únicamente se puede obtener analizando su código, y depende de cada juego.

Posteriormente, el proyecto ha sido replicado a otra máquina de las mismas características que las mencionadas anteriormente. También agradecer al grupo de investigación Graphics and Imaging Lab por dejarnos hacer uso de un ordenador Ubuntu 14.04, con una gráfica Nvidia GeForce GTX 980 Ti, donde también ha sido instalado el proyecto.

Se ha tenido acceso a un clúster de Nvidia[15] durante 4 días, que opera con Scientific Linux y dos tarjetas gráficas Nvidia Tesla K80. El proyecto ha sido replicado en esta máquina a la que ha sido accedida por ssh.

También se ha participado en el programa de Nvidia de solicitud de una tarjeta gráfica para uso en investigación. La tarjeta solicitada ha sido una Titan X, pero finalmente se ha acabado el proyecto sin recibir noticias por parte de Nvidia acerca de esta solicitud.

[15] http://www.nvidia.es/object/k80-gpu-test-drive-es.html

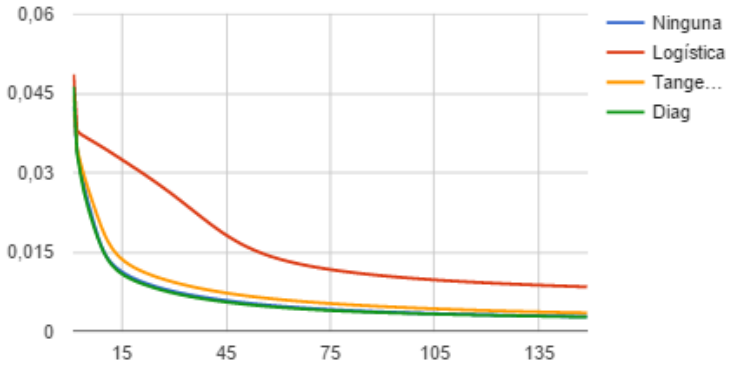
Para optimizar la memoria de la GPU, mientras un aprendizaje está en ejecución no se hace uso de la interfaz gráfica del ordenador, ya que ésta consume entorno a un 25 % de la memoria de una tarjeta gráfica de 1 GB. Esta medida permite agrandar el modelo de red neuronal hasta el usado en este proyecto, definido en la sección 4.2.

	Memoria (GB)	CUDA Cores	Op. coma flotante	#GPUs
Nvidia GeForce 550 Ti	1	192	691.2	1
Nvidia Geforce GTX 980 Ti	6	2816	5632	1
Nvidia Tesla K80	24	4992	4.061	2
Nvidia Titan X	12	3072	7.000	1

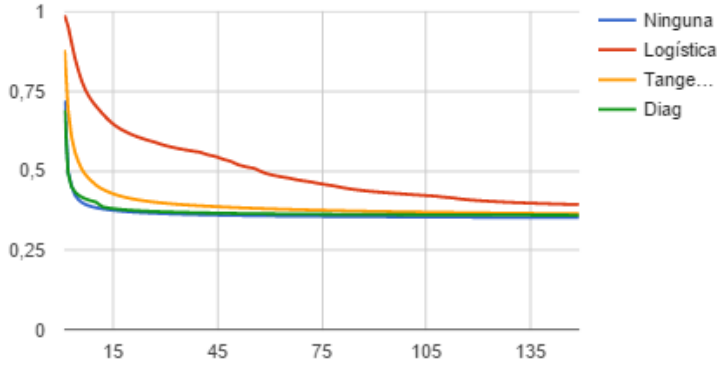
Cuadro 4: Especificaciones de las tarjetas mencionadas

C. Pruebas de entrenamiento de autocodificadores

A continuación se representa el error de reconstrucción obtenido en cada uno de los entrenamientos con autocodificadores. Estos entrenamientos utilizan diferentes arquitecturas del autocodificador, variando su número de capas convolucionales, su función de activación, el uso de dropout y la normalización de los datos de entrenamiento.

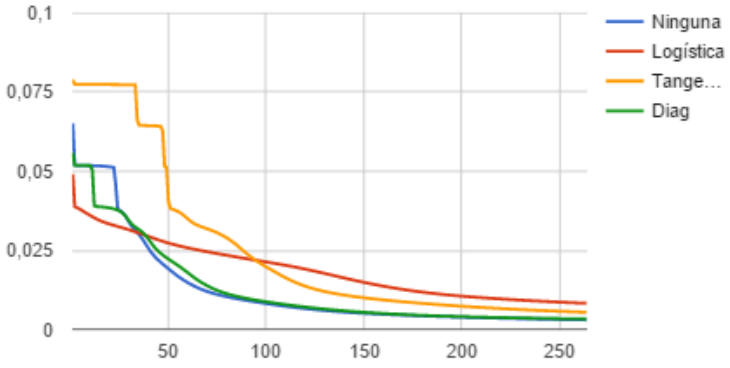


(a) Datos de entrenamiento no normalizados

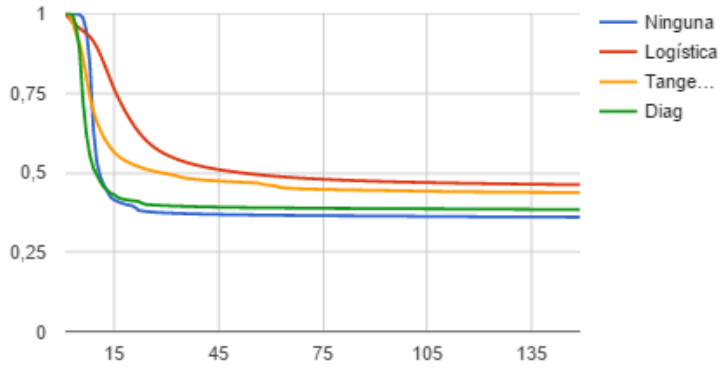


(b) Datos de entrenamiento normalizados

Figura 19: Entrenamiento de un autocodificador de una capa convolucional

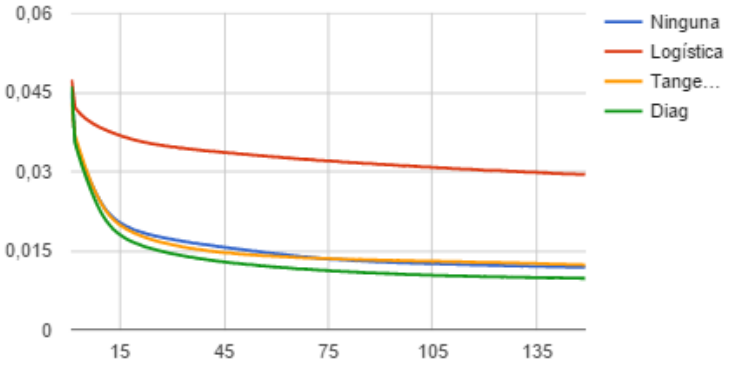


(a) Datos de entrenamiento no normalizados

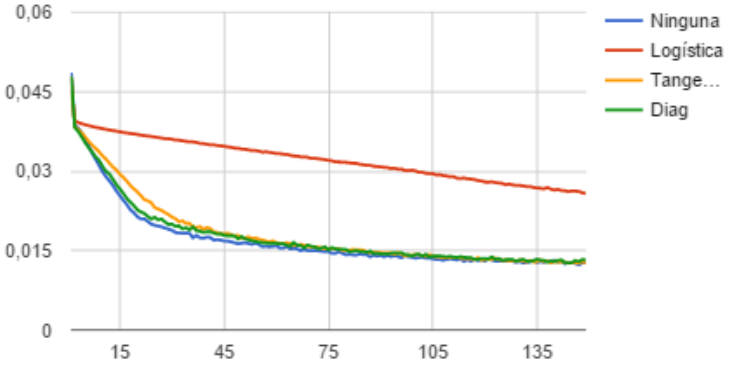


(b) Datos de entrenamiento normalizados

Figura 20: Entrenamiento de un autocodificador de dos capas convolucionales



(a) Red neuronal con dropout



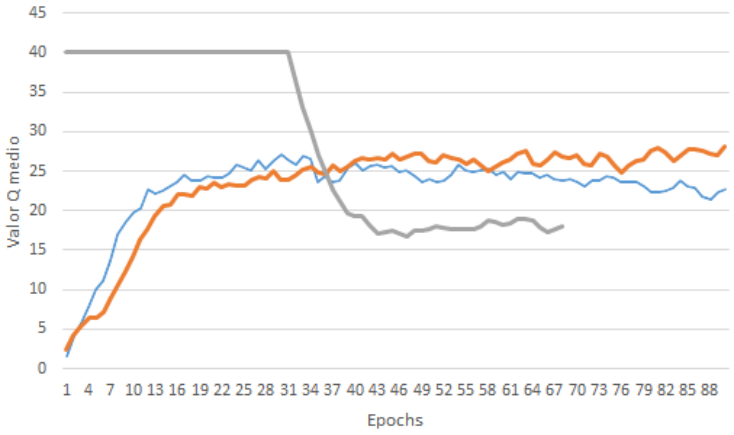
(b) Red neuronal con dropout espacial

Figura 21: Entrenamiento de un autocodificador de una capa convolucional con datos no normalizados

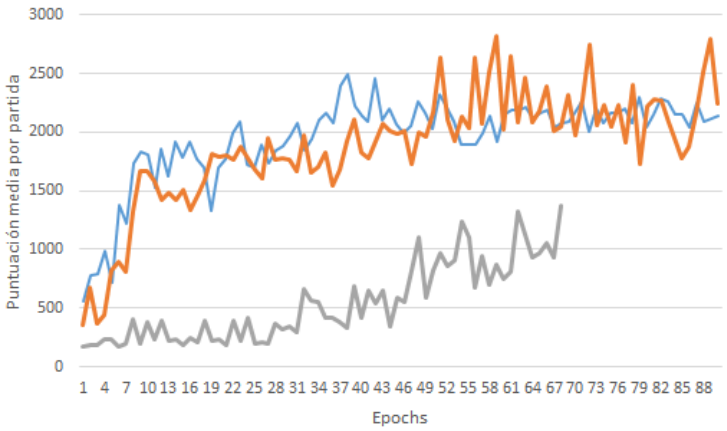
D. Resultados obtenidos del pre-entrenamiento de la red neuronal

D.1. Resultados del pre-entrenamiento no supervisado

El eje horizontal representa los epochs de entrenamiento. Cada epoch corresponde a 250.000 frames de entrenamiento, equivalentes a 4.6 horas de tiempo real de juego.



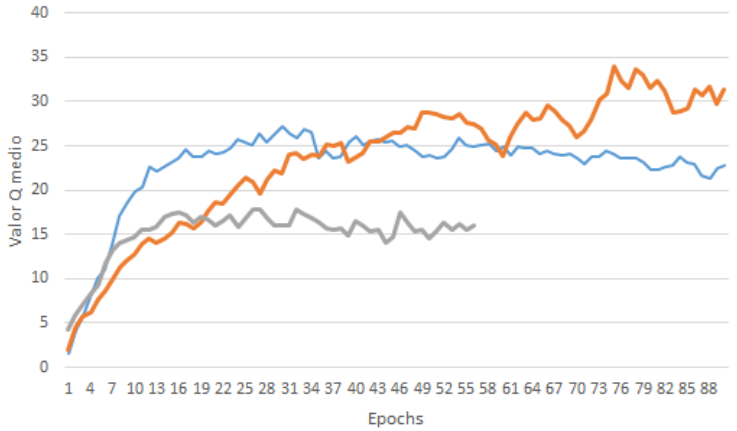
(a) Media del valor Q predicho



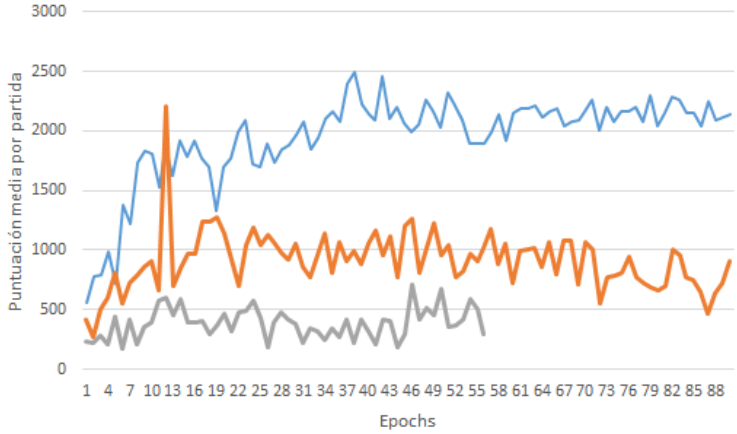
(b) Puntuación media obtenida por partida

Figura 22: Entrenamiento de Ms Pacman con las dos capas convolucionales pre-entrenadas. En azul, el entrenamiento original. En naranja, pre-entrenado con autocodificadores. En gris, pre-entrenado con K-means.

Para el pre-entrenamiento de toda la parte convolucional mediante K-means en la Figura 22, únicamente se han podido pre-entrenar 31 filtros de la primera capa y 11 de la segunda, debido a la generación de clústers vacíos. Los valores Q medios obtenidos al comienzo del aprendizaje han sido modificados para su visualización en la gráfica. Éstos comienzan con un valor muy elevado, sobre-estimando las acciones elegidas por el agente. El resultado de esta sobre-estimación se ve reflejado en la puntuación obtenida de juego durante el proceso de evaluación.

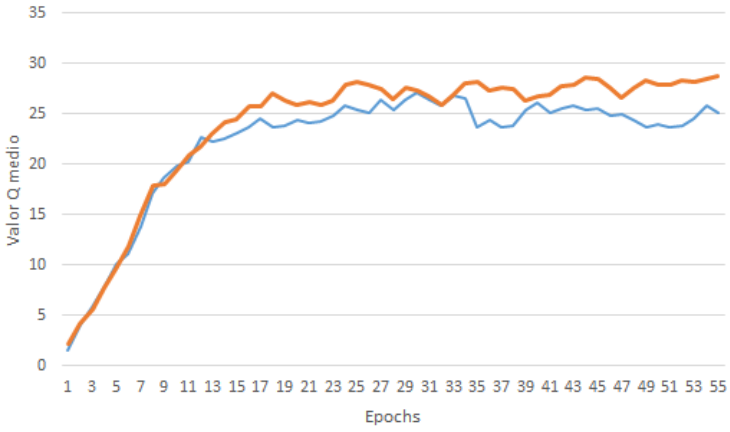


(a) Media del valor Q predicho

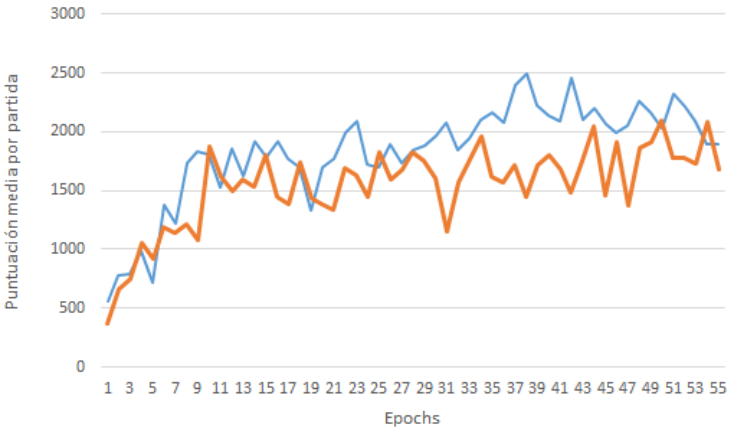


(b) Puntuación media obtenida por partida

Figura 23: Entrenamiento de Ms Pacman con una capa convolucional pre-entrenada y congelada. En azul, el entrenamiento original. En naranja, pre-entrenado con autocodificadores. En gris, pre-entrenado con K-means.



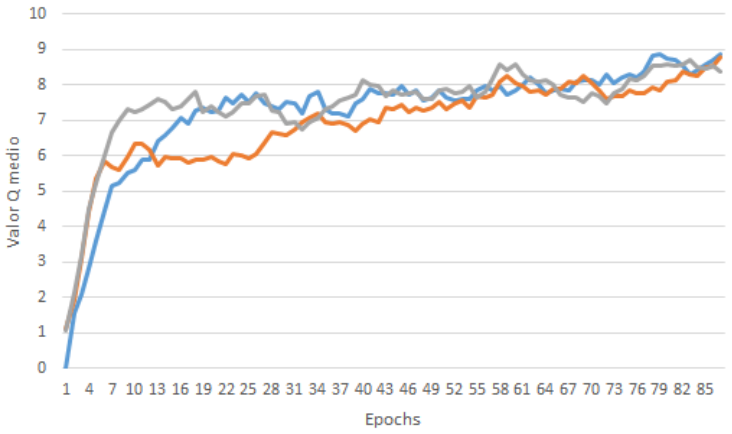
(a) Media del valor Q predicho



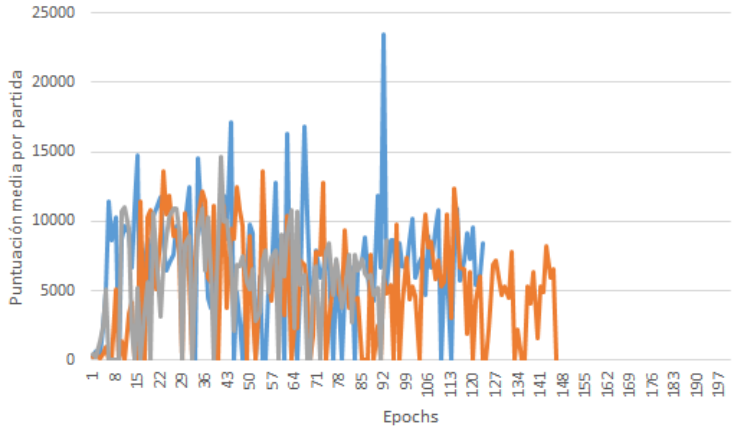
(b) Puntuación media obtenida por partida

Figura 24: Entrenamiento de Ms Pacman con 16 filtros pre-entrenados y congelados en la primera capa convolucional. En azul, el entrenamiento original. En naranja, pre-entrenado con autocodificadores.

El entrenamiento con 16 filtros congelados reflejado en la Figura 24, no ha podido ser implementado con K-means por las restricciones de tiempo y capacidad de computación. Únicamente se muestran los resultados obtenidos a partir de el pre-entrenamiento con autocodificadores.



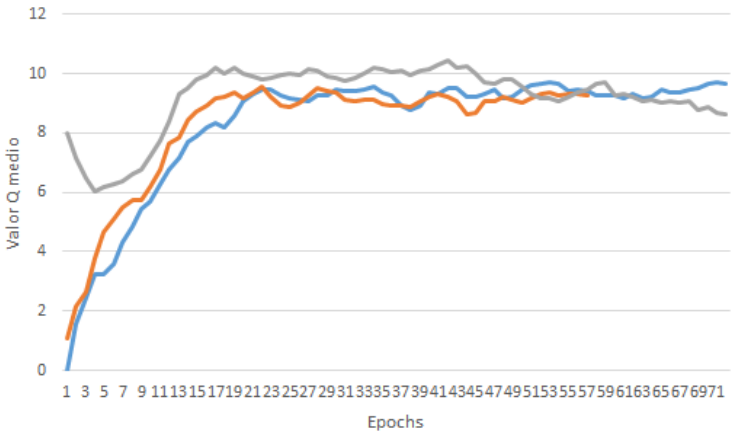
(a) Media de el valor Q predicho



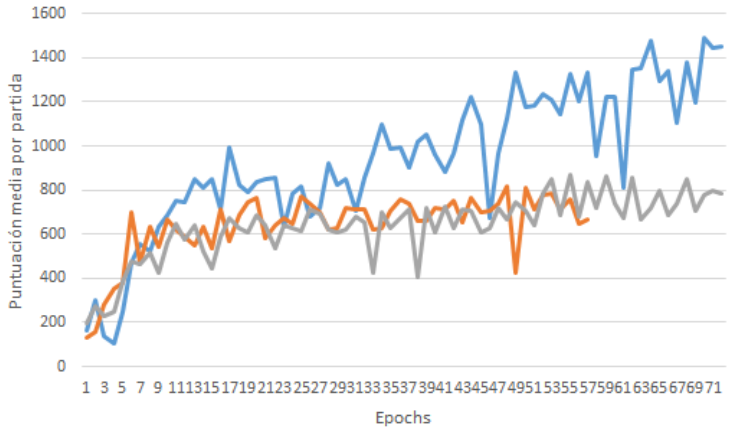
(b) Puntuación media obtenida por partida

Figura 25: Entrenamiento de Phoenix pre-entrenado con autocodificadores. En azul, el entrenamiento por defecto. En naranja, con las dos capas convolucionales pre-entrenadas. En gris, con 16 filtros de la primera capa convolucional pre-entrenados y congelados.

D.2. Resultados de la trasferencia de conocimiento

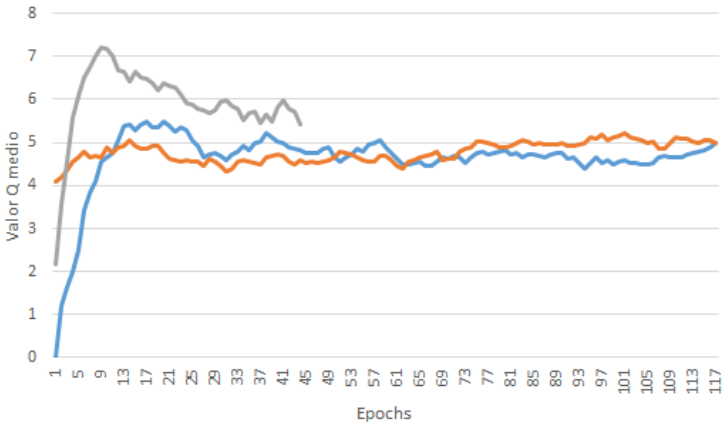


(a) Media de el valor Q predicho

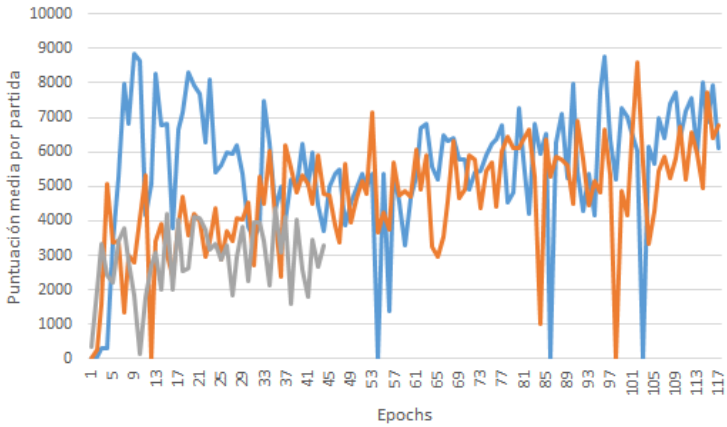


(b) Puntuación media obtenida por partida

Figura 26: Entrenamiento de Space Invaders. En azul el entrenamiento por defecto. En azul, el entrenamiento original. En naranja, entrenado con la primera capa convolucional pre-entrenada y congelada con los filtros entrenados con Demon Attack. En gris, entrenado con todos los pesos de las capas convolucionales y totalmente conectadas aprendidos para Demon Attack.



(a) Media de el valor Q predicho



(b) Puntuación media obtenida por partida

Figura 27: Entrenamiento de Demmon Attack. En azul, el entrenamiento original. En naranja, entrenamiento con todos los pesos de las capas convolucionales y totalmente conectadas aprendidos para Space Invaders. En gris, entrenamiento con las capas convolucionales pre-entrenadas con los filtros aprendidos con Phoenix.