

# Tema-5.-Aprendizaje-automatico.pdf



Anónimo



Sistemas Inteligentes



4º Grado en Ingeniería de Computadores



Facultad de Informática  
Universidad Complutense de Madrid

Formamos  
**talento** para un futuro  
Sostenible



**EOI** Escuela de  
organización  
industrial

MÁSTER EN  
**Big Data &  
Business Analytics**

[saber más](#)

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato  
→ Planes pro: más coins

## TEMA 5: APRENDIZAJE AUTOMÁTICO.

### 1. Introducción al aprendizaje automático (Machine Learning)

Aprendizaje: "El acto, proceso o experiencia de adquirir conocimiento o aptitudes".

Automático: mecanismo o aparato que funciona en todo o en parte por sí solo.

Machine Learning visto como un agente

El sistema es capaz de aprender a resolver un problema, esto sitúa a los agentes basados en Machine Learning dentro de los agentes que aprenden.

Machine Learning permite predecir el comportamiento de un sistema a futuro, podemos englobar a los agentes como agentes deliberativos o basados en objetivos, aunque no son capaces de generar un plan. Permite que los agentes puedan adaptarse a entornos cambiantes (dinámicos).

¿Cómo aprenden las máquinas?

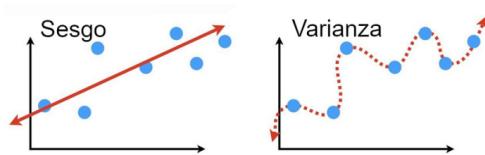
Se requiere memoria para almacenar el nuevo conocimiento y abstraer el problema para permitir enfrentarse a problemas nuevos similares.

Machine learning realiza un aprendizaje inductivo (extrae conclusiones en base a ejemplos). El algoritmo debe construir un modelo en base a los ejemplos que se le proporciona, es decir, busca un patrón en los datos de entrada. Para poder hacer esta tarea, se requiere abstraerse de los pequeños detalles que no aportan información relevante para los casos generales.

Al hacer esta abstracción se introducen sesgos para que los modelos de machine learning sean aplicables a ejemplos no usados en el entrenamiento. A esto se le conoce como tener capacidad de generalización.

Sesgo: es capaz de construir un modelo simplificado del problema, intentando extraer las características generales que los describen. El sesgo introduce un error que nos ayuda a predecir el comportamiento.

Varianza: no introducir sesgos y esto es algo que normalmente no queremos. También se le denomina sobre adaptación.

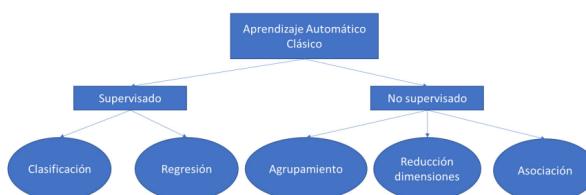


#### Tipos de aprendizaje automático.

- Aprendizaje supervisado: requieren que los ejemplos que se introducen para aprender tengan también la solución al problema.
- Aprendizaje no supervisado: se lleva a cabo sobre ejemplos que no informan del resultado esperado.
- Aprendizaje por refuerzo: la retroalimentación la obtienen del mundo exterior (del entorno en el caso de un agente). Por tanto, aprenden a base de prueba y error.

#### Tipos de problemas que se pueden abordar.

- De clasificación: identificar a cuál de un conjunto de categorías pertenece una nueva observación. Se espera una salida discreta.
- De regresión: la salida que se espera es continua.
- De agrupamiento: si no se conoce a priori el número de clases ni el tipo de estas y lo que se pretende es precisamente conocerlas.
- De reducción de dimensiones: extraer las características de unos datos.
- De asociación: se agrupan por secuencia. Pretende establecer relaciones entre variables (correlaciones)



#### Aprendizaje supervisado.

Intenta deducir una función que permita obtener a partir de unos datos de entrada la solución deseada. Tanto los datos de entrada como el valor deseado se proporcionan al sistema. Al conocer la solución, el algoritmo puede estimar cómo de lejos está de dicha solución usando el error. Cada instancia de entrenamiento debe presentarse como un vector de entrada y una salida esperada.

### Representación de los datos.

Los datos se representan en forma de matriz de  $n \times m$  siendo  $n$  el número de instancias que disponemos para entrenar el algoritmo y  $m$  el número de atributos que disponemos para cada ejemplo.

Las instancias de entrenamiento son puntos en el espacio  $m$  dimensional.

### Importancia de la correcta representación.

- Los individuos deben ser correctamente codificados, evitar en lo posible el ruido (dots con errores o con atributos que no aporten nada).
- Debe haber suficientes ejemplos como para cubrir la mayor parte del espacio del problema que se pretende resolver.
- Los datos a ser posible deben estar en la misma escala (0-1).
- Discretización de variables numéricas.
- Convertir categorías en valores binarios.

### Preparación de los datos.

Tratamiento de valores perdidos o nulos:

- Se pueden descartar las filas si son pocos.
- Calcular el punto medio en variables numéricas.
- Seleccionar el valor modal en las cualitativas.

### Capacidad de generalización.

Necesitamos que el sistema prediga la salida de instancias no usadas en el entrenamiento. Debemos evaluar al algoritmo no sólo por el error que comete con los datos de entrenamiento si no también con datos no usados en el entrenamiento.

### Cross Validation.

Cross Validation permite que el porcentaje de los datos de entrenamiento no sea siempre el mismo y se calcula el error medio de cada división realizada.

- **Error cuadrático medio** (Mean Square Error) entre el pronóstico y el real. También el error absoluto medio

$$MSE(Y, Y') = \frac{1}{N} \cdot \sum_{i=1}^N (y_i - y'_i)^2$$

- **Accuracy** o tasa de exactitud: Aciertos entre el total.

$$\text{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN} = \frac{V}{N}$$

La tasa de aciertos puede presentar problemas en casos en los que las clases estén desequilibradas

- Tasa de verdaderos positivos (**Recall**). ¿Qué proporción de positivos reales se identificó en forma correcta?

$$\text{Recall}(TVP) = \frac{VP}{P} = \frac{VP}{VP + FN}$$

### **Ejemplo**

Dada la matriz de confusión: Calcula precision, recall, F1Score.

	C observada T	C observada F
C pronosticada T	True Positive (VP)	False Positive (FP)
C pronosticada F	False Negative (FN)	True Negative (VN)

- Valor Predictivo Positivo (**Precision**). ¿Qué proporción de identificaciones positivas fue correcta?

$$\text{Precision}(VPP) = \frac{VP}{VP + FP}$$

- **F1 Score** (Mejor para elegir un modelo ya que es sólo un valor)

$$F1Score = 2 \cdot \frac{\text{Precision}(VPP) \cdot \text{Recall}(TVP)}{\text{Precision}(VPP) + \text{Recall}(TVP)} = \frac{2VP}{2VP + FP + FN}$$

$$\bullet \text{ Precision: } VPP = \frac{VP}{VP+FP} = \frac{176}{176+107} = 0,6219 = 62,19\%$$

$$\bullet \text{ Recall: } TVP = \frac{VP}{VP+FN} = \frac{176}{176+316} = 0,3577 = 35,77\%$$

$$\bullet \text{ F1Score: } F1 = 2 \cdot \frac{\text{Precision}(VPP) \cdot \text{Recall}(TVP)}{\text{Precision}(VPP) + \text{Recall}(TVP)} = 2 \cdot \frac{0,62 \cdot 0,35}{0,62 + 0,35} = 0,4488$$

### **Técnicas de aprendizaje automático supervisado.**

#### K-Nearest Neighbor o k vecinos más cercanos.

El algoritmo almacena los datos para recuperarlos en el futuro.

Cuando se presenta un nuevo caso éste es comparado con la base de casos y se extrae aquel más similar (o los  $K$  más similares) para comprobar cuál era la clase a la que pertenecía.

Se necesita definir una función de similitud entre dos ejemplos. Si la función de similitud no es capaz de medir correctamente la similitud que existe entre dos casos dados, la asunción de que a similares casos se espera que la instancia sea de la misma clase pierde vigencia y KNN deja de clasificar correctamente.

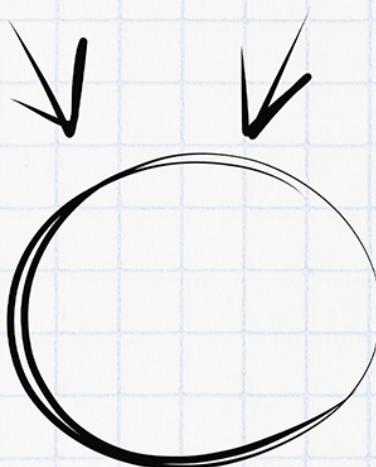
La solución a seleccionar depende del  $K$  elegido. Si  $K = 1$  será la instancia recuperada más similar. Si  $K > 1$  habrá que tomar una decisión de qué clase elegimos. Normalmente el criterio más lógico es la clase mayoritaria.

# Imagínate aprobando el examen

## Necesitas tiempo y concentración

Planes	PLAN TURBO	PLAN PRO	PLAN PRO+
diamond Descargas sin publi al mes	10 🟡	40 🟡	80 🟡
clock Elimina el video entre descargas	✓	✓	✓
folder Descarga carpetas	✗	✓	✓
download Descarga archivos grandes	✗	✓	✓
circle Visualiza apuntes online sin publi	✗	✓	✓
glasses Elimina toda la publi web	✗	✗	✓
€ Precios	Anual <input type="checkbox"/>	0,99 € / mes	3,99 € / mes
			7,99 € / mes

Ahora que puedes conseguirlo,  
¿Qué nota vas a sacar?



**WUOLAH**

# Sistemas Inteligentes



**Comparte estos flyers en tu clase y consigue más dinero y recompensas**



- 1** Imprime esta hoja
- 2** Recorta por la mitad
- 3** Coloca en un lugar visible para que tus compis puedan escanear y acceder a apuntes
- 4** Llévate dinero por cada descarga de los documentos descargados a través de tu QR

## Banco de apuntes de la



Hay multitud de **medidas de distancia** que podemos usar:

- **Distancia Euclídea:** Para cada atributo de X e Y

$$D - Euclidea(X, Y) = \sqrt[2]{\sum_{i=1}^N (x_i - y_i)^2}$$

- **Distancia de Manhattan:** Para cada atributo de X e Y

$$D - Man(X, Y) = \sum_{i=1}^N |x_i - y_i|$$

- **Distancia de Mahalanobis:** la similitud entre dos variables aleatorias multidimensionales. Se diferencia de la distancia euclídea en que tiene en cuenta la correlación entre las variables aleatorias.

Formalmente, la distancia de Mahalanobis entre dos variables aleatorias con la misma distribución de probabilidad  $\vec{x}$  e  $\vec{y}$  se calcula de la siguiente forma:

$$D - Mahalanobis(\vec{x}, \vec{y}) = \sqrt[2]{(\vec{x} - \mu)^T * C^{-1} * (\vec{x} - \mu)}$$

Donde  $\mu$  es la media de los datos y  $C^{-1}$  es la inversa de la matriz de covarianzas.

- **Distancia de Minkowski:** generalización de la euclídea o la de manhattan donde el exponente y la raíz pueden ser cualquier número.

$$D - Minkowski(X, Y) = \sqrt[p]{\sum_{i=1}^N (x_i - y_i)^p}$$

- **Distancia de edición:** el número de cambios que hay que hacer para convertir una instancia en otra. Generalización de Hamming.

- casa → cala (sustitución de 's' por 'l') = 1
- casa → calle (sustitución de s por L, sustitución a por e e inserción de l) = 3

Otra aproximación es ponderar el peso que tiene cada uno de los atributos en el cálculo de la distancia. Esto implica **introducir información del dominio** ya que debemos saber o intuir que ciertas variables aportan más información a la hora de decidir cuáles coger.

$$D - Euclidea - Ponderada(X, Y, P) = \sqrt[2]{\sum_{i=1}^N p_i(x_i - y_i)^2}$$

Donde P es el vector de pesos asociado a cada entrada.

En principio podemos ponderar cualquier distancia pero en unas tiene más sentido que en otras.

### Ventajas:

- No paramétrico (salvo que usemos distancias ponderadas). No hace suposiciones explícitas sobre la forma de funcionar de los datos, evitando los peligros de la distribución subyacente de los datos.
- Algoritmo simple tanto de explicar como de interpretar.
- Alta precisión (relativa). Es bastante alta, aunque no superior a otros modelos más sofisticados. Pero a pesar de su aparente simplicidad, si se elige correctamente la distancia puede ofrecer resultados bastante buenos
- El proceso de entrenamiento es inmediato

### Inconvenientes:

- Es muy sensible a los atributos irrelevantes. Hacer una buena selección de atributos relevantes es fundamental.
- Es sensible al ruido. Esto se puede mitigar haciendo que K sea grande ya que reduce el ruido
- La ejecución es lenta si hay muchos datos de entrenamiento, ya que tiene que procesar todos los datos. Existen métodos para optimizarlo usando partición espacial pero aún así es costoso
- Es caro en memoria ya que ocupa mucha memoria si hay muchos casos (Y tiene difícil solución, salvo limitar la memoria de trabajo)

### Árboles de decisión.

Son muy útiles para visualizar las diversas opciones de las que se dispone para resolver un problema o modelar un comportamiento. Se pueden convertir en reglas.

Un árbol de decisión está compuesto de dos tipos de nodo, un nodo condicional y una clase. Los nodos condicionales son los nodos internos del árbol y la clase, los nodos hoja o terminales.

Cada condición (nodo condición) es una pregunta para el sistema acerca de una variable y tiene dos posibilidades, que sea cierta o falsa. En función de eso seguirá un camino y otro. Los nodos finales son nodos hoja y nos dice la clase.

Algunas de las implementaciones más famosas de estos algoritmos son el ID3, J48 o C4.5 (clasificación) y M5 (regresión).

Este algoritmo utiliza la entropía (medida de incertidumbre o de desorden) para ayudar a decidir qué atributo debe ser el siguiente en ser evaluado en el árbol.

$$Entropia(s) = \sum_{i=1}^c -p_i * Log_2 p_i$$

Donde c son los posibles valores de clasificación, S es el conjunto de todos los ejemplos y  $p_i$  es la proporción de ejemplos de S que están en la clase i

Y la ganancia:

$$Ganancia(S, A) = Entropia(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} Entropia(S_v)$$

Donde  $V(A)$  es el conjunto de todos los valores posibles para el atributo A y  $S_v$  es un subconjunto de S para el cual el atributo A tiene el valor v

Como selecciona el atributo más prometedor de entre todos, podemos concluir que realiza una búsqueda voraz entre los mejores atributos. Despues aplica "Divide y Vencerás" recursivamente con el problema.

**Ventajas:**

- El entrenamiento es muy rápido.
- Es fácil de interpretar los resultados por un humano, es un algoritmo de caja blanca.
- Para algunos problemas consigue una buena precisión.
- Se pueden convertir fácilmente en reglas.
- No requiere una preparación de los datos demasiado exigente.
- Puede trabajar con variables cualitativas y cuantitativas.

**Desventajas:**

- Es muy dependiente al ruido de la entrada.
- Los árboles de decisión tienden al sobre-entrenamiento.
- No se puede garantizar que el árbol generado sea el óptimo.
- Hay conceptos que no son fácilmente aprendibles pues los árboles de decisión ya que las particiones del espacio de soluciones que puede hacer son aquellas que son representables mediante una sucesión de hiperplanos. Si no hay una aproximación lineal al problema, puede que den un modelo poco efectivo.
- Se recomienda balancear el conjunto de datos antes de entrenar.

#### Versiónes extendidas: Random Forest.

Este método ejecuta diferentes árboles de decisión y se realiza un proceso de votación para elegir cuál es la predicción final. Los RF son métodos de conjunto (ensemble methods) por este motivo. El número de árboles que genera es un parámetro del sistema:

- Dividimos nuestra serie de datos en varios subconjuntos compuestos aleatoriamente de muestras
- Se entrena un modelo en cada subconjunto
- Para cada nodo, elegir aleatoriamente m variables en las cuales basar la decisión. m debe ser < que el número total de variables
- Se combinan todos los resultados de los modelos (por votación)

Estos métodos obtienen mejores resultados en general que los árboles de decisión convencionales, aunque dificultan la comprensión del modelo generado.

**Ventajas:**

- Generalmente genera resultados muy buenos.
- Fácil de calcular
- Dar estimaciones de qué variables son importantes para clasificar

**Desventajas:**

- Sobreajusta si hay mucho ruido
- Es difícil de interpretar.

#### Support Vector Machine.

Son un conjunto de algoritmos de aprendizaje supervisado y están propiamente relacionados con problemas de clasificación y regresión.

Los vectores de soporte son los puntos que definen el margen máximo de separación del hiperplano que separa las clases. Se llaman vectores porque estos puntos tienen tantos elementos como dimensiones tenga nuestro espacio de entrada.

Hay veces en las que no hay forma de encontrar un hiperplano que permita separar dos clases. En estos casos, decimos que las clases no son linealmente separables. Para resolver este problema podemos usar un kernel. El truco del kernel consiste en inventar una dimensión nueva en la que podamos encontrar un hiperplano para separar las clases.

**Ventajas**

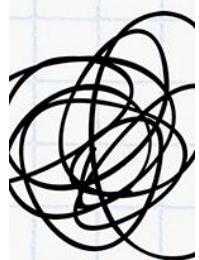
- Eficaz en espacios de grandes dimensiones.
- Todavía eficaz en casos donde el número de dimensiones es mayor que el número de muestras.

Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato  
→ Planes pro: más coins

pierdo  
espacio



Necesito  
concentración

ali ali ooooh  
esto con 1 coin me  
lo quito yo...

WUOLAH

- Utiliza un subconjunto de puntos de entrenamiento en la función de decisión (llamada vectores de soporte), por lo que también es eficiente en memoria.
- Versátil: se pueden especificar diferentes funciones del núcleo para la función de decisión. Se proporcionan kernels comunes, pero también es posible especificar kernels personalizados

Desventajas

- Si el número de características es mucho mayor que el número de muestras, evite el exceso de ajuste al elegir las funciones del Kernel y el término de regularización es crucial.
- Los SVMs no proporcionan directamente estimaciones de probabilidad, éstas se calculan utilizando una validación cruzada

## 2. Redes de neuronas

### Introducción a las redes de neuronas.

Esta tecnología se engloba dentro de la IA Subsimbólica que diseña sistemas genéricos, en este caso, intentando modelizar el funcionamiento del propio cerebro humano.

Es un sistema basado en neuronas artificiales que simulaba el funcionamiento de las neuronas biológicas.

Demostraron que cualquier función de cómputo podía calcularse sobre una red de dichas neuronas y que éstas podrían llegar a aprender.

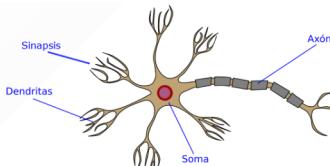
#### La neurona biológica.

La neurona se considera como la unidad funcional básica del sistema nervioso, que en los organismos vivos es el encargado de su mantenimiento vital. El sistema nervioso tiene como funciones principales el funcionamiento global del ser vivo y su relación con el entorno.

Las células nerviosas cuentan, como el resto de las células, con un cuerpo celular y un núcleo donde se encuentra la información genética de la célula o ADN. Pero además cuentan con una serie de prolongaciones con las que se intercomunican entre sí.

Partes de la neurona biológica:

- El axón es la ramificación de salida de la neurona. A través de él se propagan los impulsos nerviosos.
- Las dendritas son ramificaciones de entrada para la célula.
- Entre la dendrita y axón se encuentra la sinapsis que es un líquido semiconductor que permite circular un cierto grado del impulso.



Cuando el impulso llega a los extremos del axón, éste libera unas sustancias químicas denominadas neurotransmisores que se generan en los terminales del axón denominados botones sinápticos. Estos neurotransmisores intentan cruzar el espacio sináptico. Pero este le ofrece una resistencia dieléctrica que debe ser superada por los neurotransmisores para poder alcanzar las dendritas de la neurona receptora.

A pesar de lo que pueda parecer, la capacidad de cómputo principal de las neuronas no está en su núcleo sino en sus interconexiones, es decir, en la sinapsis.

#### Redes de neuronas artificiales.

La neurona artificial es una analogía matemática y su complejidad sináptica es mucho menor que la biológica.

Definimos "neurona artificial" como un elemento que posee un estado interno ( $S$ ) denominado nivel de activación que simula la resistencia dieléctrica de la sinapsis. Además, recibe un conjunto de señales que le permiten cambiar de estado a través de una conexión con otras neuronas, que influyen en el nivel de activación con señales positivas y negativas (simulando las neurotransmisores).

La función matemática que permite cambiar el estado de activación dependiendo de las entradas se denomina función de activación.

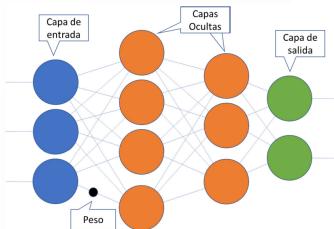
Cada entrada de la neurona es multiplicada por un peso que es modificable durante el aprendizaje. Cada peso permite que llegue una cantidad mayor o menor de la señal procedente de otra neurona.

Una red neuronal es una colección de neuronas cuya salida está conectada a las entradas de otras neuronas formando un grafo dirigido donde los vértices son las neuronas y las aristas son las conexiones entre ellas.

### Tipos de neuronas.

- Neuronas de entrada: son las encargadas de recibir los datos del exterior de la red e inician la propagación de dichos datos por toda la estructura.
- Neuronas ocultas: neuronas que permiten enlazar las neuronas de entrada y las neuronas de salida, aunque también pueden estar conectadas a otras neuronas ocultas.
- Neuronas de salida: son las que muestran al exterior los resultados obtenidos después de que la entrada haya recorrido toda la red.

Una capa es una colección de neuronas que tienen una función común.



Normalmente en cada neurona se hace la suma ponderada de los pesos para calcular su activación (aunque también se usa max y min).

Si la función de activación es lineal, añadir más capas en la red es irrelevante y se podría crear una red de una sola capa oculta equivalente. Pero si la función de activación no es lineal (lo habitual), entonces añadir más capas si tiene sentido.

### Funciones de activación más usadas.

- La función escalón donde  $\alpha$  es el valor umbral de activación:

$$y(x) = \begin{cases} 1 & \text{si } x \geq \alpha \\ -1 & \text{o} \\ 0 & \text{si } x \leq \alpha \end{cases}$$

- función sigmoidal (logística)

$$y(x) = \frac{1}{1 + e^{-xp}}$$

- La tangente hiperbólica:

$$y(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

- La función rectificadora (ReLU):

$$y(x) = \max(0, x)$$

Las funciones sigmoidal e hiperbólicas son diferenciables y monótonas y son no lineales, valores que nos interesan.

### Aprendizaje de la red.

Que la red aprenda consiste en obtener el valor de los pesos que conectan las diferentes capas que permitan aproximar lo más posible la función que queremos aprender.

El principal mecanismo de aprendizaje de una red de neuronas es el algoritmo de retropropagación que veremos a continuación como parte del perceptrón multicapa, que nos servirá de modelo de red de neuronas base.

### Perceptrón multicapa.

Usa como función de entrada la suma ponderada y como función de activación la que el usuario establezca.

El perceptrón multicapa está demostrado que es un aproximador universal de cualquier función.

### Algoritmo de retropropagación.

La idea principal que sustenta al Perceptron es su mecanismo de aprendizaje basado en la regla delta, pero generalizada a n capas ocultas.

La regla delta se basa en el error cuadrático medio que se produce en las salidas de la red si son comparadas con la salida real del dato de entrenamiento. El error de toda la red E, donde  $e(p)$  es el error producido por el patrón p:

$$E = \frac{1}{N} \sum_{p=1}^N e(P)$$

El error de un patrón  $e(p)$  es el siguiente:

$$e(p) = \frac{1}{2} \sum_{i=1}^n (d(p_i) - y(p_i))^2$$

Donde  $d(p)$  es el resultado correcto del patrón  $p$ -ésimo del conjunto de entrenamiento e  $y(p)$  es la salida de la red para dicho patrón. Es decir, que por cada una de las neuronas de salida se calcula la diferencia al cuadrado entre lo que debería haber salido y la salida real de la red.

Sea un perceptrón multicapa genérico de  $C$  capas.

$$W^{C-1,C}(p) = W^{C-1,C}(p-1) - \alpha \cdot \frac{\partial e(p)}{\partial W^{C-1,C}}$$

El aprendizaje de un peso cualquiera de las conexiones entre la última capa oculta y la de la salida es igual al valor que poseía en el instante anterior ( $p-1$ ) menos el factor de aprendizaje  $\alpha$  multiplicado por la derivada parcial del error con respecto a dicho peso.

$$W^{C-1,C}(p) = W^{C-1,C}(p-1) + \alpha \cdot \delta^C(p) \cdot a^{C-1}$$

Donde  $a^{C-1}$  es la activación de la neurona de la capa anterior (la penúltima). Y  $\delta^C(p)$  es el gradiente (derivada parcial) de la capa de salida.

Esta ecuación se puede generalizar a cualquier capa  $c$  de la siguiente forma (**Ecuación 01**):

$$W^{c,c+1}(p) = W^{c,c+1}(p-1) + \alpha \cdot \delta^{c+1}(p) \cdot a^c$$

Ahora tenemos que calcular el factor delta  $\delta$  de esta ecuación. Este factor es el que varía si es la última capa o las anteriores. Veamos el caso de la última capa. En este caso  $\delta^{c+1}(p) = \delta^C(p)$

Ahora tenemos que calcular el factor delta cuando no es la capa final. Aquí tenemos que propagar el valor del error por la red. Así pues, para calcular  $\delta^{c+1}(p)$  donde  $c+1$  indica la segunda capa que interconecta la matriz  $W^{c,c+1}(p)$  necesitamos conocer el delta de la siguiente capa (la  $c+2$ ) (**Ecuación 04**):

$$\delta^{c+1}(p) = f(y)(1-f(y)) \cdot \sum_{j=1}^{|c+2|} \delta^{c+2}(p) \cdot W_j^{c+1,c+2}$$

Es decir, para calcular por ejemplo los pesos de la matriz de pesos  $W^{1,2}$  que comunica la primera capa con la segunda capa. Necesitamos el factor  $\delta^2$  de la capa 2.

Para calcular  $\delta^C(p)$  necesitamos calcular la derivada de la función de activación. Como estamos asumiendo la sigmoidal su derivada es  $f \cdot (1-f)$ . Entonces el factor delta se calculará como sigue (**Ecuación 02**):

$$\delta^C(p) = (d(p) - y(p)) \cdot f(y)(1 - f(y))$$

Notez que  $y$  es en realidad un vector que puede tomar i valores y que depende del tamaño de la capa de salida. Por tanto  $\delta^C(p)$  también es un vector. Asumamos la capa de salida tiene  $|C|$  elementos, la ecuación para cada uno de esos elementos sería  $\forall i = 1..|C|$  (**Ecuación 03**):

$$\delta_i^C(p) = (d(p)_i - y(p)_i) \cdot f(y_i)(1 - f(y_i))$$

Para tener ese factor necesitaremos saber el error de la salida y multiplicarlo por el  $\delta^3$  es decir por el de la siguiente capa. Pero no directamente, ponderandolo con la matriz de pesos  $W^{2,3}$  para trasladar esa gradiente a la siguiente capa.

Como antes el factor  $\delta$  en realidad es un vector, por lo que para calcular cada uno de los componentes lo generalizamos de la siguiente forma:

(**Ecuación 05**):

$$\delta_i^{c+1}(p) = f(y)_i(1-f(y)_i) \cdot \sum_{j=1}^{|c+2|} \delta^{c+2}(p) \cdot W_{ij}^{c+1,c+2}$$

1. Inicializar los pesos y umbrales de la red a valores aleatorios.
2. Por cada patrón de entrada
  - a. Cargar las entradas en las neuronas de entradas.
  - b. Propagar la entrada por la red (Ecuaciones 3 y 5).
  - c. Calcular el error de dicho patrón.
  - d. Calcular todos los  $\delta$  de todas las conexiones de la red.
  - e. Calcular los pesos empezando por la salida hacia la de entrada.
  - f. Almacenar el error cometido.
3. Calcular el error promedio: repetir esto por cada ciclo de aprendizaje

### Umbrales.

Algunas redes tienen además de los pesos unos umbrales de activación que van conectados a cada neurona como si fuesen una pata más que está conectado a un valor que también puede ser aprendido.

$$u^{c+1}(p) = W^{c+1}(p-1) + \alpha \cdot \delta^{c+1}(p)$$

### Hiperparámetros de un perceptrón multicapa.

- Las capas de entrada y salida las determina el problema y la codificación elegida del mismo.
  - Capa de entrada: típicamente 1 por cada atributo de los ejemplos. Las entradas mejoran si están normalizadas a valores (0/1).
  - Capa de salida: en función de cómo codifiquemos la salida.
    - Regresión: una única salida o tantas como valores queramos predecir.
    - Clasificación: codificar la salida, una salida por clase (la mayoritaria gana) o codificación binaria.
- Las capas ocultas sí son cuestiones que decide el diseñador de la red, cuántas y qué neuronas tiene cada una.
- La función de activación.
- El factor de aprendizaje  $\alpha$ , normalmente suele funcionar con valores pequeños, pero depende del problema y puede ser dinámico.
- Número de iteraciones máximas: número de veces que se presentan los datos de entrenamiento a la red.

- Criterio de parada: si el error de validación llega a un valor con el que estamos cómodos.

#### Características importantes de la RN.

Como todos los algoritmos de ML, las redes de neuronas pueden sobreadaptarse a los datos de entrenamiento. Hay que vigilar el entrenamiento y la validación. Si el error de validación comienza a crecer mucho es recomendable parar. También puede converger prematuramente en mínimos locales y estancarse.

Son bastante buenas tolerando el ruido en la entrada pero por contra son algoritmos de caja negra y necesitan muchos datos para que el entrenamiento funcione.

#### Redes de neuronas recurrentes.

Se caracterizan por la existencia de bucles entre los nodos. Normalmente la recurrencia afecta a una parte de la red, que presenta los datos procesados por la misma como parte del proceso de entrenamiento.

Las redes de Jordan conectan las salidas como entradas de la red y las de Elman conectan una de las neuronas ocultas como entrada de la red.

Esto permite realimentar la red y son ideales cuando el valor depende de valores previos.

#### ¿Cómo encontramos los parámetros correctos?

Existen diferentes técnicas como:

- Grid search
- Random search
- Usando algoritmos genéticos.
- Algoritmos de descenso de gradiente.

Sólo podemos explorar una pequeña parte del espacio de los hiperparámetros en un tiempo razonable.

#### ¿Cuándo parar una red?

Epoch se refiere a utilizar una vez todo el conjunto de entrenamiento. Suele haber un número máximo.

Se puede terminar el proceso automáticamente cuando no se producen mejoras del entrenamiento durante un cierto número de iteraciones.

### **DeepLearning**

El algoritmo de aprendizaje basado en gradiente tiene un problema fundamental y es que cada una de las capas aprende a diferentes velocidades. Las capas más cercanas a la salida les afecta más que a las primeras. De esta forma, necesitamos otra forma de funcionar para permitir aumentar la capacidad de representación de la red sin que el gradiente del error se vuelva inestable.

Se ha demostrado que una red con una sola capa oculta puede modelar funciones muy complejas, siempre que tenga suficientes neuronas.

Las redes profundas tienen una eficiencia de parámetros mucho más alta que las superficiales, es decir, pueden modelar funciones complejas utilizando exponencialmente menos neuronas que redes poco profundas.

En cuanto a las capas ocultas, una práctica común es dimensionarlas para formar un embudo, es decir, la capa  $i$  tendrá (bastantes) más neuronas que la capa siguiente  $i + 1$ . Esta aproximación se basa en que muchas características de bajo nivel pueden unirse en muchas menos características de alto nivel. Las capas iniciales trabajan con muchas características de bajo nivel, mientras que las capas ocultas cercanas a la capa de salida, trabajan con menos características de más alto nivel.

Las funciones de activación, se suele usar la función de activación ReLU en las capas ocultas (o una de sus variantes). Para la capa de salida, la función de activación softmax es generalmente una buena opción para tareas de clasificación cuando las clases son mutuamente excluyentes. Cuando no son mutuamente excluyentes (o cuando solo hay dos clases), generalmente se prefiere la función logística.

#### Softmax

Son neuronas que normalmente se colocan en la capa de salida y que recogen las entradas de la red de una forma similar a como hace la función sigmoide, sumando ponderadamente los valores de la capa previa a la capa actual.

Esta función permite interpretar la salida como una probabilidad. Cada una de las salidas de la red es una clase y su valor la probabilidad de que esa sea la clase de los datos.

#### Autoencoders.

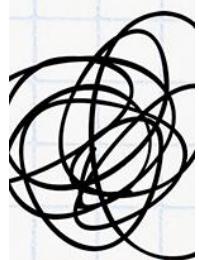
Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato

→ Planes pro: más coins

pierdo  
espacio

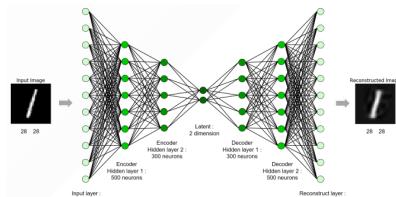


Necesito  
concentración

ali ali ooooh  
esto con 1 coin me  
lo quito yo...

WUOLAH

Son un tipo especial de redes neuronales que funcionan intentando reproducir los datos de entrada en la salida de la red.



Lo interesante de este tipo de redes es que permite extraer las características más relevantes de una red usando aprendizaje no supervisado. No suelen utilizar neuronas Softmax en la capa de salida, si no sigmoidal o Relu.

Los autoencoders pueden tener múltiples capas ocultas (autoencoders apilados o stacked).

El entrenamiento de las redes se realiza de forma análoga a las tradicionales.

Si hay muchas capas, se pueden entrenar poco a poco (capa a capa) para simplificar el entrenamiento.

#### Convolutional networks

Las redes convolucionales son un tipo especial de redes neuronales para procesar datos con tipología cuadrículada, como las imágenes.

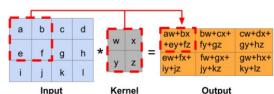
#### Reducir las dimensiones de la imagen

Por el problema anterior del tamaño de los datos de las imágenes necesitamos reducir la complejidad de la imagen, pero conservando su información esencial.

La convolución es una operación lineal sobre dos funciones que producen una tercera que expresa como la forma de una es modificada por la otra.

Hasta ahora conectábamos todas las neuronas de la entrada con la oculta (lo cual multiplicaba exponencialmente los pesos). Las redes convolucionales hacen un mapeo de una capa a la siguiente, reduciendo el tamaño y sin interconectar todos los nodos.

En el ejemplo anterior aplicamos un kernel 2x2 a una matriz de 4x4, convirtiendo dicha matriz en una matriz 3x3 que es una combinación lineal de los anteriores valores.

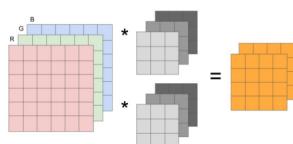


Cada capa oculta detecta la misma característica, pero esta detección la realiza por toda la capa previa. Los pesos compartidos y el sesgo compartido conforman el kernel.

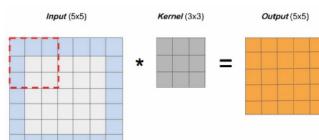
Las redes convolucionales deberán utilizar múltiples mapas de características según el número de patrones a detectar en los datos de entrada. La capa oculta por tanto puede estar compuesta de múltiples mapas de características que se van extrayendo con cada uno de los kernels.

#### Partes de una CNN

- Kernel: reduce la dimensionalidad de la imagen. Si la imagen tiene varios canales se pueden sumar los canales aplicando un kernel por cada canal (RGB) y combinándolos en una solo. También podemos tener más de un kernel para extraer características diferentes.



- Padding: añadir 0s para mantener la dimensionalidad de la matriz de salida.



- Step: consiste en no aplicar la convolución a toda la matriz, y saltarse pasos.

#### Otras capas

- Capa de agrupamiento (pooling): selecciona el valor máximo del conjunto de valores de entrada.
- Capa totalmente conectada: para clasificar.
- Capa ReLu: aplicar no linealidad.
- Capa de dropout: desactiva un número de entradas poniéndolas a 0 para evitar el sobreentrenamiento.

Aplicaciones: visión artificial en coches autónomos, reconocimiento facial, clasificación de imágenes, transferencia de estilos...

### 3. Aprendizaje no supervisado

#### Introducción.

Si no se conocen las clases o valores esperados de los datos, tenemos que recurrir al aprendizaje supervisado.

#### Reducción de dimensionalidad.

Los datos con alta dimensionalidad (gran número de atributos) son difíciles de interpretar y procesar. En cambio, los de baja dimensionalidad son más fáciles de procesar.

#### Principal Component Analysis (PCA).

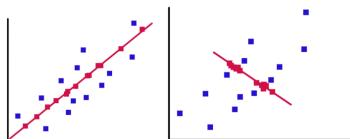
PCA nos permite reducir la dimensionalidad de los datos de entrada sin perder toda la información y transforma las variables correlacionadas en número menor de variables no correlacionadas. Esto se hace proyectando (producto punto) los datos originales en el espacio reducido utilizando la descomposición en valores y vectores propios de la matriz de covarianza/correlación.

- PCA proyecta la información a un espacio de menor dimensión.
- PCA es efectivo cuando la correlación es alta. Podemos mirar primero la correlación y después analizar si podría ser útil o no aplicar PCA.

Veamos un ejemplo: Vamos a imaginar un problema de 2 dimensiones por simplicidad que queremos convertir en 1 dimensión



Ahora queremos proyectar estos puntos a 1 dimensión. ¿Cómo lo hacemos? Podemos hacerlo de dos formas:



La primera gráfica tiene mucha más información que la segunda (mayor varianza)

PCA debe obtener la mayor varianza de los datos proyectados, manteniendo las variables implicadas como independientes entre sí.

Existen dos formas de calcularlo:

- Método basado en la matriz de correlación: datos dimensionalmente no homogéneos.
- Método basado en la matriz de covarianzas: cuando los datos son dimensionalmente homogéneos.

#### Método basado en correlaciones.

El método parte de la matriz de correlaciones. Para cada uno de los n individuos del conjunto de entrenamiento con m variables de entradas construimos la matriz de correlación:

$$R = |r_{ij}| = \frac{\text{cov}(F_i, F_j)}{\sqrt{\text{var}(F_i) \cdot \text{var}(F_j)}}$$

Los valores propios de la diagonal deben sumar m. Estos m valores propios son los pesos y cada uno de los principales puede ser descrito como una combinación lineal de los otros.

#### Método basado en las covarianzas.

El objetivo es transformar un conjunto dado de datos de dimensión n x m a otro conjunto de datos de menor dimensión n x l con la menor pérdida de información útil posible utilizando para ellos la matriz de covarianza ( $l \leq \min(n, m)$ ).

$$\text{cov}(X, Y) = \frac{(X_i - \hat{X})(Y_i - \hat{Y})}{n}$$

#### Análisis del algoritmo de reducción de dimensionalidad:

- Presupone que los datos pueden ser proyectados usando una combinación lineal.
- Asume cierta correlación entre las características.

- Se pierde información.
- Se pierde interpretabilidad de los datos. Ahora los datos con los que se entrena el modelo de ML son atributos que están creados a partir de combinaciones lineales de los atributos originales.
- Problema si hay muchos casos atípicos.

### Aplicaciones:

- Mejora el desempeño de los algoritmos de ML
- Reducción del ruido (al haber menos atributos y con más información).
- Mejora la visualización (si tengo más de tres atributos no puedo visualizarlo).

### Técnicas de agrupamiento o Clustering.

El objetivo es agrupar los  $n$  individuos de nuestro conjunto de datos en una serie de grupos de forma que:

- Los individuos del mismo grupo sean los más similares entre sí.
- Los individuos de grupos diferentes sean los más diferentes entre sí.

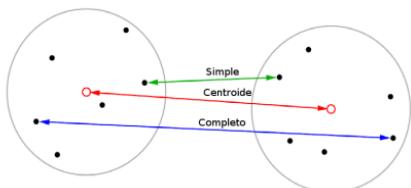
Estos grupos nos revelarán cierta estructura de los datos. Por lo que puede ser útil hacer clustering aunque sólo sea para analizarlos.

- Cuántos grupos puede y cuáles son más numerosos.
- Grupos más homogéneos o más dispersos, atípicos, etc.

### Distancia entre clusters.

Usamos la similitud y hay diferentes formas de medirla:

- Entre los centroides del cluster.
- Enlace simple: entre los puntos más próximos de los clusters.
- Enlace completo: entre los puntos más alejados de los clusters.
- Distancia media.



### Estrategia aglomerativa.

Este algoritmo inicialmente crea una clase por cada ejemplo de entrada. Después y siguiendo el criterio de similitud elegida, busca los dos conceptos más cercanos entre sí y los agrupa en un concepto ficticio, sacando estos dos nodos de la lista y sustituyéndolos por el nuevo concepto. Este nuevo nodo tendrá a los antiguos como hijos suyos.

El proceso se repite iterativamente hasta que no quedan más nodos que agrupar.

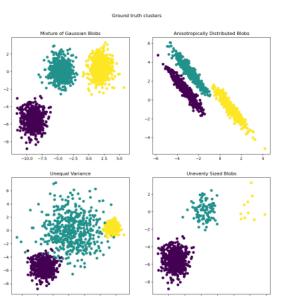
### K-Means

La idea es maximizar la similitud entre los elementos de una clase y minimizar la similitud entre las diferentes clases. No crea jerarquías de clases si no clases de un único nivel.

El algoritmo elige un conjunto de  $k$  semillas, siendo  $k$  un parámetro dado por el usuario. Después iterativamente va agrupando los ejemplos en las  $k$  clases y recalculando el valor de dicha semilla, como centroide de dicha clase.

Debe existir un criterio de parada o convergencia que hace que el algoritmo se detenga. Un posible criterio puede ser si durante dos ciclos no cambia la clasificación de los ejemplos.

### **Ejemplo de clustering con K-means**



### Métodos para intuir el valor óptimo de K.

Si K es más grande que la distancia acumulada de los ejemplos de cada cluster a los centroides será siempre menor. Por lo que debemos intentar es encontrar el valor de K que produce un punto de inflexión en la curva. A esto se le denomina técnica del Codo y se puede hacer manualmente graficando la función de "inercia" de k-means para cada k elegido.

#### Coeficiente de Silhouette para medir el rendimiento.

El coeficiente de Silhouette mide qué tan cerca está tan cerca una muestra a las otras muestras de su cluster y qué tan lejos está con respecto a las muestras del cluster más cercano. Este coeficiente toma valores de [-1, 1], -1 sería si los clusters están superpuestos, 0 si hay overlap y 1 que no se tocan.

El puntaje de Silhouette es el promedio de los coeficientes de Silhouette de todas las samples y se computa con las clases. Valores bajos cercanos a 0 nos indica que la cantidad de los clusters no es buena.

## 4. Aprendizaje por refuerzo

El aprendizaje por refuerzo es un tipo especial de aprendizaje automático que se realiza de forma online. Es decir, no se entrena previamente si no que aprende en base a prueba y error.

Hay una cierta supervisión del resultado obtenido ya que un evaluador le indica si su comportamiento es correcto, pero no le indica el resultado correcto.

Si se tiene una buena medida del rendimiento del agente, puede re-entrenar el agente si el entorno cambia, permitiendo adaptarse mejor al entorno.

#### Proceso de decisión de Markov.

Este tipo de problemas juega con decisiones probabilísticas y decisiones no deterministas. Está principalmente asociado a lo que se denominan problemas de decisión de Markov (MDP) donde:

- El entorno se encuentra en un conjunto de posibles estados S.
- Existe una función de transición  $T(s,a)$  que es desconocida de forma que dado un estado s del entorno y una acción a ejecutada en ese estado, el entorno cambia a un estado s'.
- Una función de refuerzo  $R(s,a)$ (normalmente desconocida), pero que el entorno informa al agente.

El objetivo es encontrar una política, preferentemente la óptima, que permita seleccionar en cada estado s aquellas acciones que maximicen en el futuro el refuerzo.

#### **Q-Learning.**

Es el algoritmo clásico de aprendizaje por refuerzo más conocido y que se basa en la definición anterior de proceso de Markov.

El agente cuando toma la decisión inicialmente no sabe la consecuencia de la misma, hasta que aprende la política óptima a través de la exploración y la experiencia, utilizando la función de valor para estimar la recompensa esperada en cada estado y acción posible.

#### Razonamiento de Q-Learning.

Se busca una política que maximice el refuerzo esperado en el tiempo (la suma de todos los refuerzos recibidos).

Si conocieramos las reglas del entorno no haría falta inferir la tabla, pero como no lo sabemos, ésta se calcula haciendo programación dinámica.

Hay que explorar el espacio de estados para conseguir encontrar la política óptima, por tanto necesitamos repetir el proceso de aprendizaje no siempre tomando las decisiones aprendidas, para adquirir más conocimiento del entorno.

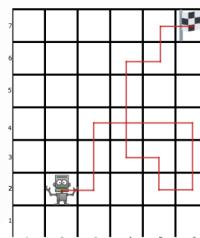
### Ejemplo

Supongamos un robot que se mueve en un entorno bidimensional desconocido y que debe aprender el camino para llegar a la salida. Las acciones del robot serán:

- MoveUp, MoveDown, MoveLeft, MoveRight
- Se le proporciona un refuerzo de 1 cuando llega a la meta, 0 en el resto. Si hubiese zonas que hacen caer al robot recibirían un -1. El mapa para el ejemplo es una cuadrícula de 6x7 casillas.

Supongamos que el comportamiento sea determinista  $\alpha = 1$  y  $\gamma = 0.8$ . Inicialmente  $e = 0$  (siempre se moverá aleatoriamente)

Supongamos el siguiente movimiento aleatorio realizado por el robot:



Importante

Puedo eliminar la publi de este documento con 1 coin

¿Cómo consigo coins? → Plan Turbo: barato  
→ Planes pro: más coins

pierdo  
espacio



Necesito  
concentración

ali ali ooooh  
esto con 1 coin me  
lo quito yo...

WUOLAH

Veamos como va cambiando la tabla Q en cada iteración.

Inicialmente  $Q = 0$

Inicialmente todos los refuerzos serán cero menos el de la casilla  $Q(5,7)$  que será:

$$Q_1((5,7), RG) = R((5,7), RG) + 0.8 \max_b(Q_0((6,7), b)) = 1 + 0.8 \cdot 0 = 1$$

Se seguirá actualizando la tabla Q con la siguiente iteración:

$$Q_2((5,6), UP) = R((5,6), UP) + \max_b(Q((5,7), b)) = 0 + 0.8 \cdot 1 = 0.8$$

$$Q_3((4,6), UP) = R((4,6), UP) + \max_b(Q((5,6), b)) = 0 + 0.8 \cdot 0.8 = 0.64$$

Y así sucesivamente hasta completar la matriz Q

La matriz Q tendrá una entrada por cada estado y cada acción como la siguiente:

s	UP	DW	LF	RG	r
6,7	0	0	0	0	0
5,7	0	0	0	1	0
5,6	0.8	0	0	0	0
4,6	0.64	0	0	0	0

### Y una vez entrenado

El agente se guía por la política Q. Para cada estado habrá una acción con más refuerzo. Esa será la que elija. Si tiene un mismo refuerzo máximo para más de una acción, entonces elegirá una al azar.

El algoritmo puede estar entrenando constantemente modificando la tabla Q en cada iteración para adaptarse a los cambios del entorno.

#### Limitaciones de Qlearning.

- El espacio de estados debe ser finito.
- Si es infinito o muy grande es muy complicado que la búsqueda encuentre una política óptima. La solución es discretizar el espacio de estados y comprimirlo.
- Aún así, es muy probable que si el problema es complejo QLearning no encuentre una solución medianamente óptima. Posibles soluciones pueden ser aprender la tabla Q con aprendizaje supervisado o hacer una aproximación mixta de aprendizaje supervisado y online.

#### Aprendizaje por refuerzo profundo.

Las limitaciones de Qlearning vienen del espacio de búsqueda y de la capacidad de representación. La tabla Q es demasiado grande para ser calculada cuando el problema crece en tamaño. Para ello existen dos aproximaciones que podemos realizar:

- La primera es aprender la tabla Q usando una red neuronal y entrenándola con ejemplos.
  - Se utiliza una red de neuronas separada para cada acción, cuya entrada es la lista de valores del estado, y cuya salida es valor de refuerzo de la acción.
- La segunda es aprender la tabla Q sin un modelo. Deep Q-Network (DQN) utiliza esta aproximación y sustituye la tabla Q por una red neuronal profunda que predice el valor de Q para todos los posibles estados.

#### ¿Cómo se entrena la red?

Entrenamos usando la ecuación de Bellman, calculamos los valores estimados y luego consideramos que tenemos un problema de aprendizaje supervisado, entrenando la red con la solución obtenida por la ecuación de Bellman.

#### Experience Replay

Evita tener demasiada correlación entre los datos de entrenamiento.

En vez de entrenar con los valores predichos por el algoritmo, lo que hacen es meterlos en un buffer y de ahí extraer un conjunto aleatorio de entrenamiento. Esto reduce la dependencia de los datos para entrenar la red. El buffer contiene una colección de tuplas de experiencia anteriores del agente. Las tuplas se agregan gradualmente al buffer a medida que el agente interactúa con el entorno.

#### Target Network.

Las estimaciones de Q-Learning se hacen en base a otras estimaciones y los estados muchas veces son muy similares. Esto hace que cambios pequeños pueden tener efectos colaterales en la red. Para reducirlo, lo que se hace es tener una red idéntica de respaldo y es con esta red con la que se obtiene los valores de !, es decir, con la que se entrena la red principal. Esta red auxiliar se va actualizando con el entrenamiento de la principal, pero no constantemente si no cada cierto tiempo.

