Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 8: Agentes inteligentes: Deep Reinforcement Learning

Recordatorios:

- Hay que entregar el laboratorio de forma individual o en grupo el día 14 de junio (fecha limite).
- Estaré atento al foro para ayudaros con las dudas pero no os voy a resolver yo el laboratorio.
- *Los últimos temas ya son teóricos, así que seguramente haga partes practicas de recuerdo en las ultimas clases de cara al examen.*

Que vamos a ver hoy…

- Reinforcement Learning
- Procesos de decisión de Markov
- *Deep Q-Learning*

Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 8.1: Reinforcement Learning

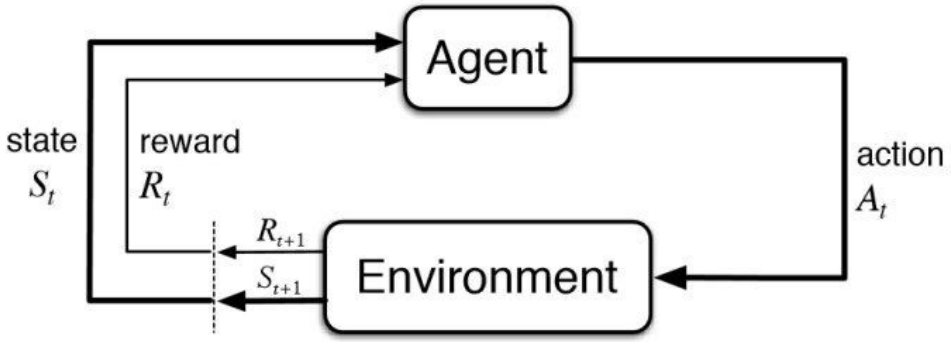
Tipos de aprendizaje

- **Aprendizaje supervisado**. Tenemos una serie de datos *x* y una salida *y* a predecir. Aprendemos una función capaz de llevarnos de *x* a *y*, como una red neuronal. Ejemplos:
  - Clasificación de imágenes
  - Clasificación de texto
  - Regresión
- **Aprendizaje no supervisado**. Tenemos una serie de datos *x*, sin valor de salida a predecir. El objetivo es aprender una estructura existente en los datos. Ejemplos:
  - Clustering
  - Density estimation
  - Reducción de dimensionalidad

Fuente de la imagen: Stanford CS231N

Reinforcement learning

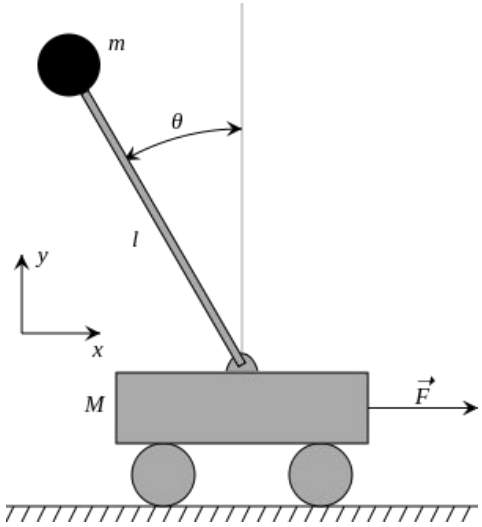
- **Aprendizaje por refuerzo (reinforcement learning)**
  - Tenemos un **agente** interactuando con un **entorno** (environment).
  - El agente se encuentra en un **estado** s y lleva a cabo **acciones** a.
  - Al realizar una acción, el agente se mueve a un **nuevo estado** y recibe una **recompensa** (reward).



Fuente de la imagen: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

Ejemplos de reinforcement learning

- Cart-pole o péndulo invertido. El objetivo es mantener el péndulo en equilibrio encima del carro.
  - El estado viene dado por el ángulo del péndulo, su velocidad angular, la posición del carro y la velocidad del carro.
  - Las acciones son empujar el carro hacia un lado u otro.
  - La recompensa es un valor positivo por cada instante de tiempo t en el que el péndulo esté en estado vertical.



Fuente de la imagen: : https://en.wikipedia.org/wiki/Inverted_pendulum

Ejemplos de reinforcement learning

- Videojuegos (por ejemplo, Doom o Starcraft). El objetivo es ganar la partida.

    - El estado viene dado por lo que el jugador ve en la pantalla (todos los píxeles en cada instante de tiempo).
    - Las acciones son el movimiento en todas las direcciones y disparar.
    - El agente es recompensado cuando elimina a un oponente y es castigado (recompensa negativa) cuando muere.

Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 8.2: Procesos de decision de Markov

Procesos de decisión de Markov

- Los **Procesos de decisión de Markov (Markov Decision Processes, MDP)** son la formulación matemática de los problemas de aprendizaje por refuerzo. Estos procesos se definen mediante la tupla:

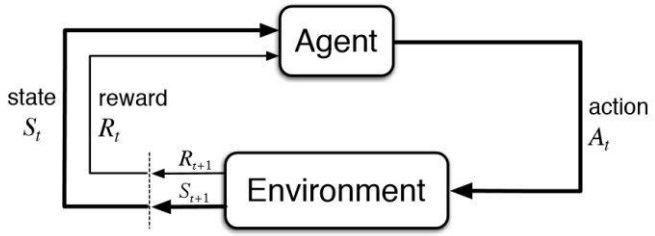
$$(S, A, P_{sa}, \gamma, R)$$

  - S es el conjunto de **estados**.
  - A es el conjunto de **acciones**.
  - P_{sa} son las probabilidades de **transiciones** entre estados. Modela una distribución de probabilidad del estado al que vamos desde el estado s al ejecutar la acción a. Define por tanto las transiciones entre estados.
  - γ es el **discount factor** ($0 < \gamma \leq 1$). Representa la importancia de obtener rewards lo más pronto posible.
  - R es la **reward function**. Modela las recompensas dadas a partir de un estado y una acción.

Propiedad de Markov

- Los procesos de decisión de Markov siguen la **propiedad de Markov**, que dice que el **estado actual caracteriza completamente el estado del entorno**. Esto implica que, a la hora de realizar una acción a_t en un estado s_t, la transición al nuevo estado al que vamos es independiente de los estados anteriores en los que podía encontrarse el entorno.

- Funcionamiento de un proceso de decisión de Markov:
  - Se empieza en un time step inicial t_0 y en un estado inicial s_0
  - Mientras el estado actual s_t no sea un estado final:
    - El agente selecciona una acción a_t
    - El entorno otorga una recompensa r_t a partir de s_t y a_t
    - Se obtiene un nuevo estado s_{t+1} según la distribución de probabilidad P_{sa} a partir de s_t y a_t



Optimal policy

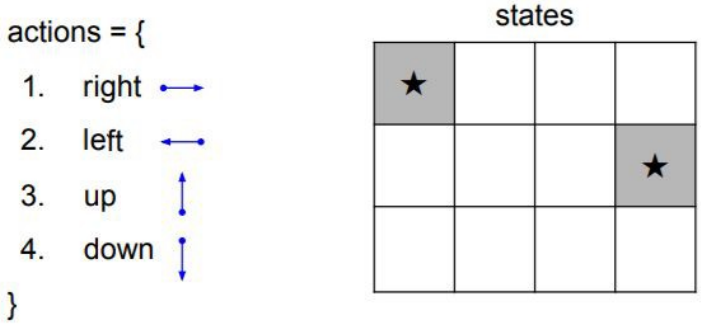
- El agente elige acciones con base a una **policy** o **política** π. La policy determina, por cada estado de S, la acción de A a realizar.

- El objetivo es encontrar la política óptima π^* que maximice la suma cumulativa de recompensas a lo largo del tiempo

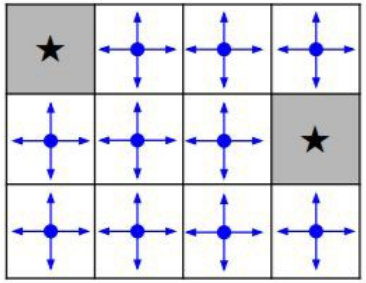
$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$$

  - Las recompensas vienen dadas por r_t.
  - Si $\gamma = 1$, no hay descuento
  - Cuanto más cercano a 0 sea γ, menos sumarán las recompensas futuras.

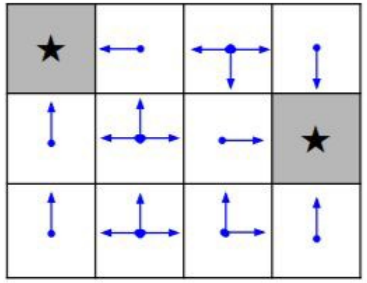
Optimal policy



- Objetivo: llegar a ★ en el menor número de acciones.
- Reward function: cada paso dado tiene r = -1



Random Policy



Optimal Policy

Fuente de la imagen:
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf

Value function

- Seguir una policy produce una trayectoria de estados, acciones y rewards:

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2...$$

- Necesitamos una forma de evaluar cómo de bueno es un estado y una acción a partir de la trayectoria potencial que podemos tomar.

- La **value function $V^{\pi}(s)$** define, para el estado s y una policy π, la recompensa acumulada esperada a partir de seguir la policy π desde el estado s.

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

- Nos permite saber cómo de bueno es un estado con una policy.

Q-value function

- La **Q-value function $Q^{\pi}(s, a)$** nos da la recompensa acumulada esperada al elegir la acción a en el estado s y, después, seguir la policy π

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

- La **optimal Q-value function Q*(s, a)** es el máximo valor de recompensas alcanzable con una policy π a partir de un estado s y una acción a.

$$Q^{*}(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

- Conociendo Q*, la policy óptima π^* viene dada por tomar la acción que tiene un valor mayor de Q* para el estado en el que estamos.

- El objetivo por tanto será aprender Q*.

Ecuación de Bellman

- La optimal Q-value function Q* satisface la **ecuación de Bellman**:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

- De manera intuitiva, esta ecuación nos dice que, partiendo de todas las acciones que podemos tomar a partir de ejecutar a en el estado s, si conocemos el valor óptimo para cada posible acción a' en el estado s' en el que hemos acabado, hemos de tomar la que maximice el valor.

Ecuación de Bellman - Value iteration

- La ecuación de Bellman nos da el primer método para obtener Q*, conocido como **value iteration**. Consiste en utilizar la ecuación de Bellman de forma iterativa:

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$

de manera que *Q* converge a *Q** cuando *i* tiende a infinito.

- Este método ha sido muy utilizado históricamente. Sin embargo, tiene el problema de que necesita calcular Q(s,a) para cada pareja (s, a) dentro de S y A. Esto hace que sólo sea tratable computacionalmente para problemas con un número de estados y acciones reducidos.

- Por ejemplo, si queremos que un agente aprenda a jugar a videojuegos a partir de imágenes, el estado viene dado por todos los píxeles de la pantalla, lo cual hace que haya un número de estados demasiado grande.

Sistemas Cognitivos Artificiales

Roberto Casado Vara

Tema 8.3: Deep Q-Learning

Deep Q-Learning

- Hasta ahora hemos visto un marco más tradicional de reinforcement learning.
- En este apartado, vamos a ver la utilización de redes neuronales profundas para aprender la función Q*.
- En particular, veremos cómo **Deep Q-Learning** fue utilizado por la empresa *DeepMind* para conseguir que agentes aprendieran a jugar a juegos clásicos de Atari **directamente a partir de las imágenes del juego**.

Deep Q-Learning

- **Problema** con algoritmos como value iteration: necesitamos calcular Q(s,a) para cada pareja de estado y acción, lo cual se vuelve intratable.

- **Solución**: utilizar una aproximación en forma de función para estimar la *Q-function*.

$$Q(s, a; \theta) \approx Q^*(s, a)$$

- De este modo, no es necesario tratar el problema de manera discreta por cada estado y acción.

- **Idea clave** del Deep Q-Learning: utilizar una **red neuronal profunda como función de aproximación**.

Deep Q-Learning

- Nuestra red neuronal calculará Q(s, a; θ), donde:
  - El input de la red es el estado s.
  - El output de la red es un Q-value por cada posible acción a.
  - θ son los parámetros de la red.

- Para aprender esta red neuronal, definimos, como de costumbre, una función de pérdida o loss function.

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

- Esta función nos dice lo bien que aproxima la red Q(s, a; θ) el valor real de los Q-values y_i.

- **Problema**: y_i es el valor real de Q* que buscamos obtener…¡es desconocido!
- **Solución**: Se estima y_i utilizando la ecuación de Bellman a partir de la red neuronal actual que está siendo entrenada.

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a')|s,a\right] \qquad \textbf{Ecuación de Bellman}$$

$$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a\right] \qquad \textbf{Cálculo de } y_i$$

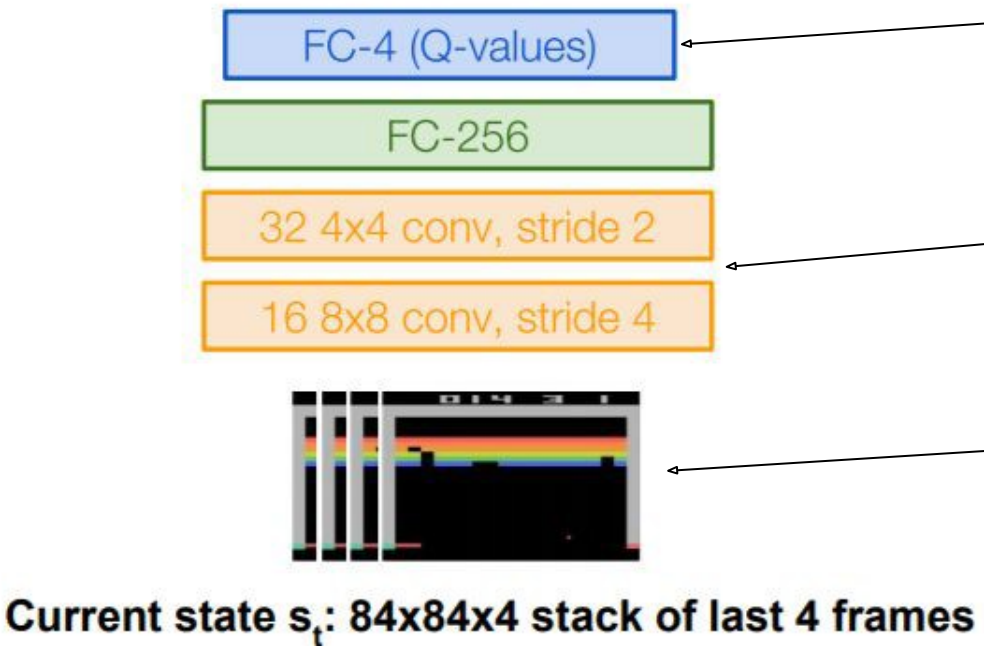
El entrenamiento se convierte en un proceso iterativo de manera que los Q-values calculados por la red neuronal se acercan a los valores que deberían de tener según la ecuación de Bellman si Q fuese óptimo.

Atari Breakout

- **Objetivo**: romper los ladrillos de la pantalla usando una bolita que controlamos con una paleta, evitando que esta caiga al vacío. A más ladrillos rotos, mayor puntuación.

- Estado: píxeles de la pantalla.
- Acciones: mover la paleta a la izquierda y a la derecha, o no moverla.
- Rewards:
  - Puntos positivos por romper ladrillos.
  - Puntuación negativa según pasa el tiempo (incentiva al agente a terminar cuanto antes la pantalla).

Atari Breakout: *Q-network*



Output: Fully connected layer con Q-value de cada acción posible.

Capas convolucionales para obtener una representación de lo que se ve en pantalla en forma de features de alto nivel.

Input: Estado. Píxeles de los cuatro últimos *frames* del juego para mayor información.

Fuente de la imagen:
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf

Deep Q-Learning para juegos: algoritmo

- Entrenar algoritmos de reinforcement learning es un proceso complejo y requiere en ocasiones de trucos y construcciones prácticas.
- Para el caso de juegos, DeepMind utilizó una idea conocida como *Experience Replay*. En vez de ir jugando y alimentar la red con inputs de juego consecutivos, se crea una memoria de experiencias conocidas (acciones, transiciones y recompensas) y se crean mini-batches de manera aleatoria a partir de ella.
- Esto evita tener *samples* de juego correlacionadas y hace el entrenamiento mucho más efectivo.

Deep Q-Learning para juegos: algoritmo

Algorithm 1 Deep Q-learning with Experience Replay

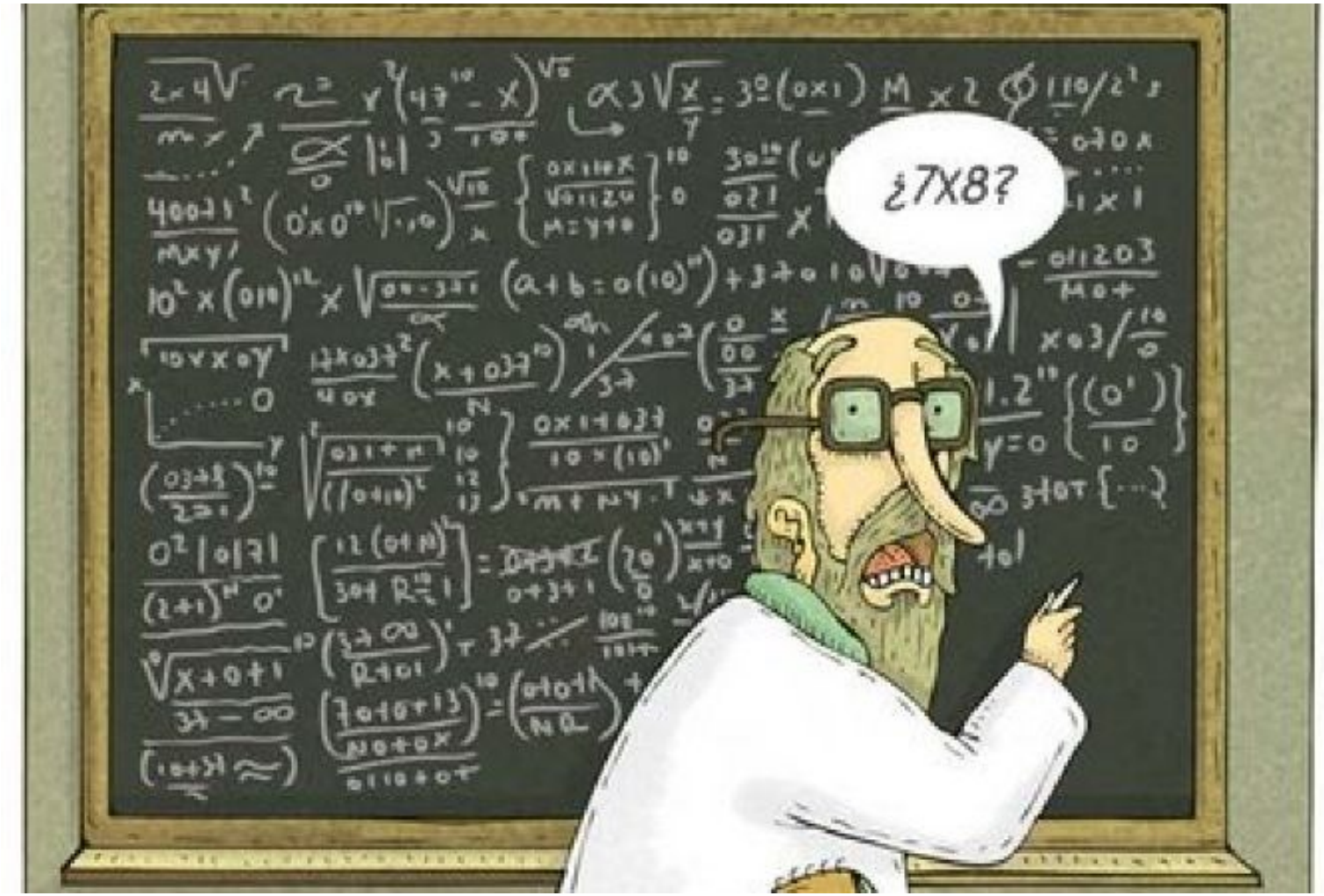
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

- Se utiliza un **emulador** del juego que permite ejecutar acciones y observar los resultados.
- *Exploration* vs *exploitation*: al jugar, en ocasiones se utiliza la función Q aprendida hasta el momento para obtener la acción a realizar (*exploitation*) y en ocasiones se eligen acciones aleatorias para explorar nuevos caminos de juego (*exploration*). Aparte de que en ocasiones queremos explorar nuevas trayectorias, al comienzo del entrenamiento la Q dada por la red neuronal es aleatoria, por lo que el algoritmo no aprendería si no hubiera exploración.

La semana que viene:

- Empezamos el tema 9 que trata de las redes neuronales en los entornos Big data.
- Acordaros de la entrega del laboratorio para el día 30 y que estaré disponible en el foro para dudas sobre el laboratorio.
- *Ya quedan pocas semanas de clase, os recomiendo que vayáis mirando los temas y aprovechéis las ultimas clases para las dudas que os puedan surgir de cara al examen final.*

¿Dudas?



www.unir.net