## 1. Objetivo del laboratorio

*Desarrollar de forma autónoma distintas implementaciones de **redes neuronales de aprendizaje supervisado** que permitan resolver distintos casos de uso. La práctica comenzará con la implementación más sencilla del funcionamiento de una neurona sencilla y terminará con la construcción de un **MLP** capaz de hacer predicciones para un caso concreto planteado.*

## 2. Elementos a utilizar:

- Lenguaje Python
- Librerías: numérica *NumPy*, estructuras de datos *pandas*, gráfica *Matplotlib* (opcional si se quieren implementar gráficas) y redes neuronales *Keras* y *Tensorflow*.
- Entorno Anaconda
- Editor Jupyter
- Archivos .csv proporcionados

## 3. Práctica 1 (Fundamentos de redes neuronales)

### Objetivo

Implementación de una red neuronal para resolver las funciones OR y XOR. La implementación se hará paso a paso, correspondiendo con los distintos apartados de la práctica. Como es necesario probar y testear el código según se desarrolla, utiliza los archivos propuestos en el apartado "**Implementación**" de esta práctica y responde a las preguntas que se plantean en "**Cuestiones**".

Debes de usar comentarios con profusión, incluyendo celdas específicas donde expliques los algoritmos que usas y el código que has programas. Cuantos más comentarios haya, mejor será la evaluación y, probablemente, menos preguntas serán necesarias en la defensa de la práctica.

### Implementación

Crea el notebook *L2P1-RRNN.ipynb*. El programa en Python deberá responder a los siguientes puntos:

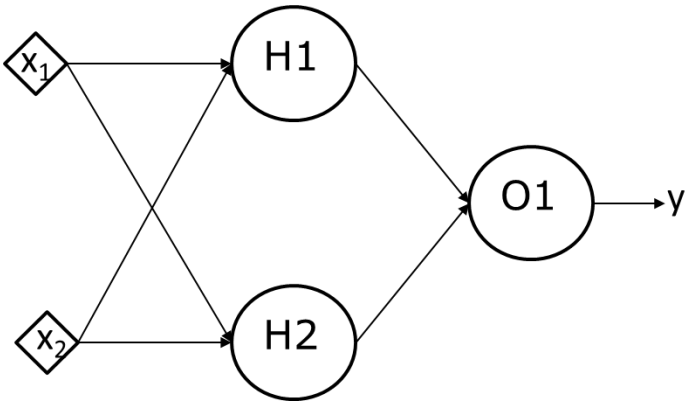


**Figura 1. Red neuronal para resolver los apartados 1 y 2**

1. **Función de activación f(S).** Con el esquema de red neuronal de la figura 1 y los datos de los ficheros *'red_neuronal1.csv'* y *'red_neuronal2.csv'* probar las funciones de activación rectifier, también conocida como **ReLU** (unidad lineal rectificada por sus siglas en inglés) y **sigmoide**. Implementa ambas funciones de activación usando funciones de Python. Utiliza las librerías *Pandas* (métodos *read_csv* y *loc*) y *NumPy* para obtener el resultado de la salida, incluido el cálculo de la función sigmoide. Crea la función calcular que realice el cálculo de la red neuronal:
   *def calcular (DataFrame valores, str funcion_activacion).*

Los datos que se pasan a la función serán:
- los valores de entrada de la red neuronal junto con los pesos de cada neurona (en una estructura de datos de tipo DataFrame)
- el nombre de la función de activación (como un string).

Imprime por pantalla cada salida de las neuronas (H1, H2 y O1) para cada caso.

2. **Función de error**. Usar la función *calcular* del apartado anterior para probar cuatro configuraciones diferentes de matrices de pesos con los datos de entrada de los ficheros: '*rn1_relu.csv*', '*rn2_relu.csv*', '*rn1_sigm.csv*' y '*rn2_sigm csv*'. Como indican los nombres de los ficheros, para los dos primeros se usará la función de activación **ReLU** y para los dos segundos **sigmoide**. A continuación, estudia si las predicciones que hacen son correctas mediante el cálculo del error de cada una de ellas. El valor esperado para las redes neuronales que usan la función de activación ReLU es 1 y en el caso de la sigmoide es 70. Escribir los valores finales para cada caso.

3. **Aprendizaje**. Con lo practicado en los dos ejercicios anteriores, implementa un PERCEPTRÓN cuyos pesos iniciales tendrán un valor al azar entre -1.0 y 1.0 ambos incluidos y que se entrenará usando la Ley de Hebb según las expresiones vistas en clase.
   - Resuelve la función **OR**. Prueba diferentes *learning rates* y *umbrales* para saber cuál es el óptimo. Sacar por pantalla los valores de salida de la red neuronal y de la función de error para cada iteración.
   - Repetir el ejercicio anterior para la función **XOR**.

   Entregar los resultados de ambas pruebas en el archivo *L2P1_apartado3.csv* y en la memoria en forma de dos tablas que incluyan para cada prueba los siguientes datos:

| Iteración | Entradas | | Pesos iniciales | | Yr | Yd | Error | Pesos finales | |
|---|---|---|---|---|---|---|---|---|---|
| | X1 | X2 | W1 | W2 | | | | W1 | W2 |

### Cuestiones

Elabora una memoria de la práctica en la que respondas a las siguientes cuestiones

1. Teniendo en cuenta los valores de salida del apartado 2 ¿Podemos saber cuál es la red neuronal que mejor predice? ¿Qué modelo predice mejor para la función 'ReLU'? ¿Y para la 'sigmoide'?
2. Recoge los resultados de las dos pruebas del apartado 3 en sendas tablas y discútelas en función de lo visto en teoría. Para los casos en los que sea posible, escribe de forma explícita la ecuación del hiperplano que se ha generado.

## 4. Práctica 2 (MLP con Keras)

### Objetivo

Utiliza la librería Keras para construir y entrenar un MLP que resuelva problemas no lineales. En vez de usar la Regla Delta Generalizada, usaremos **Adam** como función de modificación de matriz de pesos (optimizer) de la forma que se indica en el apartado de "**Implementación**". Responde a las preguntas que se plantean en "**Cuestiones**".

### Implementación

Crea el notebook *L2P2-MLP.ipynb*. El programa en Python deberá responder a los siguientes puntos:

1. Usando la API de Keras, crea un modelo neuronal "*fully connected*" y *secuencial*. El modelo tendrá que resolver la función XOR. Prueba con varios números de neuronas en la capa oculta y la cantidad de iteraciones. Hacer al menos 3 modelos intentando optimizar los resultados. Usar como optimizer '*Adam*', funciones de activación '*ReLU*' y '*softmax*', y función de error '*Mean Squared Error*'.

   Nota: Usa arrays de numpy para los valores de entrenamiento y sus salidas de predicción.

API de Keras: https://keras.io/
Adam: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

2. Usar los valores *'p2_entrenamiento.csv'* y *'p2_objetivos.csv'* para crear un modelo. Entrenarlo con distintos valores de '*learning rates'* con valores 0,0000001, 0,01, 0,1 y 1 (usa un array de Numpy). Usar el 'optimizer', las funciones de activación y la función de error del apartado anterior.

   Nota: El 'learning rate' es un atributo de la clase 'optimizer'.

### Cuestiones

Continúa en la memoria respondiendo a las siguientes cuestiones

1. Crea una tabla con los resultados del MLP que resuelve la función XOR que incluya el número de neuronas de la capa oculta, el número de epochs, los valores de salida y el último valor error para dicho learning rate. Cada fila de la tabla será una de las variaciones anteriores. Crear otro modelo cambiando las funciones de activación.

| Neuronas capa oculta | Epochs | Salida | Error |
|---|---|---|---|

2. Crea una tabla con los resultados del MLP que resuelve el ejercicio 2 que incluya el número de neuronas de la capa oculta, el número de epochs, el learning rate y el último valor error para dicho learning rate. Cada fila de la tabla será una de las variaciones anteriores. Crear otro modelo cambiando las funciones de activación.

| Neuronas capa oculta | Epochs | Learning rate | Error |
|---|---|---|---|

## 5. Práctica 3 (Juego de Tronos con Keras)

### Objetivo

Utiliza lo practicado en este laboratorio para resolver un caso práctico "real" usando la librería Keras y una arquitectura MLP para predecir la probabilidad de muerte de los personajes de **Juego de Tronos** (GoT)

### Implementación

Crea el notebook *L2P3-GoT.ipynb*. El programa en Python deberá responder a los siguientes puntos:

3. Crea un MLP y entrénalo con la información del archivo '*got.csv'*. Los valores que se usarán para entrenar serán:
   - los que indican el género del personaje
   - si aparece en los libros 1, 2, 3, 4 y 5
   - si está casado
   - si es noble o no
   - el número de personas de su entorno que han muerto
   - si es un personaje popular.

   El valor que nos indica la posibilidad de sobrevivir de un personaje es '**alive'**, con valores entre 0 y 1.
2. Una vez creada la red neuronal y viendo que puede predecir correctamente, usar los valores del archivo '*got_predicciones.csv'* y predice cual es el personaje con más posibilidades de morir.

### Cuestiones

Continúa en la memoria respondiendo a las siguientes cuestiones

1. Dibuja la arquitectura de red y escribe el '*optimizer'* y las *funciones de activación* y de *error* usadas. Indica cual es el valor de error final y el valor de *alive* para todos los personajes. ¿Quién es el personaje con más posibilidades de morir?

2. Cambia el fichero *predicciones_got.csv* para que sea otro el personaje con más probabilidades de morir y compruébalo. ¿Qué cambio has realizado? Justifícalo desde el punto de vista de las redes neuronales. ¿Todas las entradas son igual de relevantes? ¿Cómo lo puedes demostrar?

## 6. Forma de entrega del laboratorio:

La entrega consistirá en un fichero comprimido RAR con nombre LAB02-GRUPOxx.RAR subido a la tarea **LAB2** que **contenga únicamente**

1. **Por cada práctica** un notebook de Jupyter (archivos con extensión **.ipynb**).
2. Una **memoria del laboratorio** en Word.

**Las entregas que no se ajusten exactamente a esta norma NO SERÁN EVALUADAS.**

## 7. Rúbrica de la Práctica:

**1. IMPLEMENTACIÓN: Multiplica la nota del trabajo por 0/1**

Siendo una práctica de IA, todos los aspectos de programación se dan por supuesto. La implementación será:

- Original: Código fuente no copiado de internet. Grupos con igual código fuente serán suspendidos
- Correcta: Los algoritmos SOM están correctamente programados. El programa funciona y ejecuta correctamente todo lo planteado en el apartado "*Cuestiones*" de cada práctica.
- Comentada: Inclusión (**obligatoria**) de comentarios.

**2. MEMORIA DEL LABORATORIO**

Obligatorio redacción clara y correcta ortográfica/gramaticalmente con la siguiente estructura:

- *Portada con el nombre de los componentes del grupo y el número del grupo*
- *Índice*
- *Resultados de la Práctica 1*
- *Resultados de la Práctica 2*
- *Discusión general de la práctica*
- *Bibliografía*

Calificación de las cuestiones:

| PRÁCTICA | CUESTIÓN | VALORACIÓN (sobre 10) |
|---|---|---|
| Práctica 1 | Cuestión 1 | 0.5 |
| | Cuestión 2 | 1,5 |
| Práctica 2 | Cuestión 1 | 1 |
| | Cuestión 2 | 2 |
| Práctica 3 | Cuestión 1 | 2 |
| | Cuestión 2 | 3 |