**Redes Neuronales.**

**Objetivos**

Implementar modelos simples de Redes Neuronales con la librería Keras.

**Preparación**

Desde una terminal de Anaconda ubicarse en el directorio de la práctica donde se encuentra los archivos con extensión “.ipynb” (notebooks) y escribir: “jupyter notebook”. Desde el entorno web, acceder a la dirección del servidor (generalmente <http://localhost:8888/>) para abrir los archivos.

**Ejercicio 1.** Utilizando los siguientes sitios, interactuá con los modelos para observar cómo las redes generan representaciones de los datos automáticamente en cada capa. El objetivo es entender de manera más intuitiva cómo la red genera su salida y puede aproximar funciones arbitrarias con suficientes neuronas.

* [Paso forward en una red de dos capas](https://ml4a.github.io/demos/simple_forward_pass/): Para comprender mejor el paso forward, en esta visualización se utiliza un diagrama donde los valores son nodos, y los pesos son aristas.
* [Red para regresión 1D](https://cs.stanford.edu/people/karpathy/convnetjs/demo/regression.html): El modelo se entrena constantemente, adaptandose a los datos nuevos que se agregan. Permite observar la salida de cada neurona de cada capa para un conjunto de datos. Prestar atención a las salidas individuales se combinan para representar finalmente a la función. Consejo: Bajar la cantidad de neuronas por capa para entender mejor sus salidas.
* [Red para clasificación 2d](https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html): Lo mismo que antes pero para un problema de clasificación con 2 variables de entrada. Observar a la derecha los valores de cada neurona de la red y sus colores. A medida que avanzan las capas, la representación de cada clase se va separando
* [Tensorflow Playground](https://playground.tensorflow.org/): Similar al caso anterior, pero ahora la visualización de las capas se hace más clara porque se hace en un espacio 2D.
* [Visualización 3D de clasificación de imágenes](https://www.cs.ryerson.ca/~aharley/vis/fc/): Observa como la imagen se va recodificando en cada capa hasta la softmax final que indica las probabilidades de clase. Nota: no se muestra, pero el primer paso de la red es convertir la imagen, que es una matriz de 28x28 valores en un vector de 28\*28=784 valores, simplemente poniendo los valores de la matriz uno al lado del otro (como en una operación *reshape* de NumPy)
* [The grand tour of network visualization](https://distill.pub/2020/grand-tour/) (opcional, largo): Visualización de varios aspectos de la codificación de los ejemplos por la red durante el entrenamiento y al final del mismo.

**Ejercicio 2** Entrená un modelo de red neuronal para regresión, en este caso para **predecir** el **consumo de nafta** (*MPG*, por *miles per gallon*) en base a los **caballos de fuerza** (Horsepower) de un conjunto de datos de automóviles. Como podrás observar, el conjunto de datos tiene una distribución no lineal respecto del consumo.

1. Verificá que el modelo propuesto inicialmente puede modelar la no linealidad
2. Intercambiá las funciones de activación de las dos primeras capas. ¿Puede aprender ahora la red?
3. Decrementa a 1 la cantidad de neuronas de las dos capas. ¿Puede aprender ahora la red? Probá volviendo a 2 algunas de esas capas ¿y ahora? ¿por qué sucede esto?
4. Incrementa a valores mayores a 10 la cantidad de neuronas de las capas 1 y 2. ¿Puede aprender a modelar el problema? ¿Varía cualitativamente la solución?
5. Volvé a utilizar 2 neuronas para las primeras capas, pero modificá la tasa de aprendizaje a 0.1. ¿Qué sucede con el modelo final?
6. Idem 5, pero con 10 neuronas.
7. Volvé a utilizar 2 neuronas para las primeras capas y una tasa de aprendizaje de 0.001, pero ahora usá un tamaño de lote (batch\_size) de 1. ¿Cómo cambia la curva del error a través de las iteraciones?
8. Editar el código para dejar una sola capa lineal sin activaciones en el modelo, de modo que se transforme en un modelo de regresión lineal. ¿Empeora el MSE/MAE en este caso?
9. Editar el código de carga de datos para que se tengan en cuenta más variable

***Archivo*: Regresión con Redes Neuronales con Keras.ipynb**

**Ejercicio 3** Entrená modelos para clasificar los conjuntos de datos de la tabla utilizando redes neuronales con topología secuencial, es decir, un modelo secuencial con varias capas.

1. Probá variando la cantidad de capas y la función de activación de cada una. Anotá los mejores resultados y la topología que utilizaste.  
   Podés escribir la topología de una red de, por ejemplo, tres capas, donde la primera tiene activación ReLU y 3 salidas, la segunda TanH y 5 salidas, y la tercera softmax y 2 salidas como:  
    Dense(3,’relu’) - Dense(5,’tanh’) - Dense(2,’softmax’).
2. Compará los resultados con los de regresión logística, en especial para los datasets 2D. ¿En qué casos mejora utilizar redes? ¿En qué casos el modelo es similar?
3. Probá utilizando una red de 5, 10 o 20 capas Dense, peso sin funciones de activación. ¿Funcionan mejor que la regresión logística estas redes? ¿Por qué?
4. Probá cambiando el tamaño de lote con los valores 4 , 16 y 64, y la tasa de aprendizaje entre 0.001, 0.01, 0.1 ¿en qué sentido cambian las curvas de error durante el entrenamiento? En particular, ¿en qué casos es suave la curva de error, y en qué casos parece tener ruido?

***Archivo*: Redes Neuronales con Keras y varios conjuntos de datos.ipynb**

C*onjuntos de datos*:

| Archivo | Mejor accuracy | Mejor loss | Iteraciones | Tasa de aprendizaje | Topología |
| --- | --- | --- | --- | --- | --- |
| 2\_clases\_simple.csv |  |  |  |  |  |
| 6\_clases\_dificil.csv |  |  |  |  |  |
| circulos.csv |  |  |  |  |  |
| diabetes.csv |  |  |  |  |  |
| iris.csv |  |  |  |  |  |
| moons\_mis-scaled.csv |  |  |  |  |  |
| moons.csv |  |  |  |  |  |

| utils.py | |
| --- | --- |
| X\_train, X\_test, Y\_train, Y\_test = dividir\_train\_test(X, Y, test\_size) | Divide el dataset de forma aleatoria en dos conjuntos. El parámetro “test\_ size” es un número entre 0 y 1 que indica el porcentaje a utilizar para el conjunto de evaluación. |
| plot\_clasificador\_2D(model, X\_train, Y\_train, X\_test, Y\_test) | Grafica en 2D el dataset junto con la frontera de decisión del modelo. Al pasar el conjunto de testeo se visualizan los dos conjuntos con diferentes marcadores. |
| plot\_training\_curves(history , acc=True) | Grafica las curvas de entrenamiento (loss) para el training y testing set. Si el parámetro “acc” es True grafica también la curva para el Accuracy. |
| plot\_confusion\_matrix(y\_true, y\_pred) | Grafica la matriz de confusión para los vectores de clases pasados como argumento. |
| print\_classification\_report(y\_true, y\_pred) | Imprime en consola un reporte del accuracy para los vectores de clases pasados como argumento. Si la clasificación es binaria se imprime además el precision, recall y f-measure. |
| plot\_tradeoff\_curves(modelo, x, y) | Grafica las curvas ROC y precision-recall para el modelo entrenado y para el conjunto pasado como argumento. |

**Ejercicio 4 (opcional).** En algunos problemas, se desea darle mayor importancia a minimizar los errores de algunos ejemplos.

Por ejemplo, para diagnosticar una enfermedad, es mejor que el modelo tenga una tasa muy baja de falsos negativos (*recall* alto), aunque la tasa de falsos positivos sea alta, ya que de no detectar la enfermedad las consecuencias son muy dramáticas. Entonces, se le dará más importancia a que se clasifiquen bien los ejemplos que tienen la enfermedad.

En otros dominios puede pasar lo contrario. Por ejemplo, en la detección de objetos astronómicos, donde la cantidad de datos es enorme, se busca automatizar la búsqueda de los mismos para ahorrar trabajo manual. Como para cada tipo de objeto hay varias instancias que se pueden observar, importa que la tasa de falsos positivos sea chica, aunque la de falsos negativos sea grande (*precision* alto).

En un problema de 2 clases, por ejemplo, el error puede escribirse como:

E = c1 E1 + c2 E2

donde E1 y E2 son los errores promedio de cada clase, y c1 y c2  son los pesos de cada clase. En general, los ci se eligen de forma que sumen a 1. Entonces, si se desea darle el doble de importancia los errores de la clase 1, se pueden elegir c1=0.66 y c2=0.33, y entonces:

E = 0.66 E1 + 0.33 E2

1. Realizá el mismo procedimiento que el ejercicio anterior, pero para los datasets descriptos a continuación. Recordá que las redes neuronales son sensibles a la escala de los datos. Intentá encontrar un balance adecuado entre precision y recall utilizando el parámetro **class\_weight** del método [fit](https://keras.io/api/models/model_training_apis/).
2. En los conjuntos de Diabetes y Ecoli, ¿cuál es el *precision* más alto que podés obtener, manteniendo el recall cercano a 1?

| Titanic.csv | Información sobre diferentes pasajeros del Titanic indicando si sobrevivió o no al viaje. |
| --- | --- |
| Diabetes.csv | Dataset con información sobre pacientes, indicando si cada uno tiene o no diabetes gestacional. |
| Ecoli.csv | Información de proteínas de pacientes, indicando si presenta o no una enfermedad. |

**Ejercicio 4 (Avanzado, Opcional).** Realiza las guías 13 a 17 de [*simplenn*](https://github.com/facundoq/simplenn)