Álgebra Lineal Computacional

2do Cuatrimestre 2025

Trabajo Práctico: Red Neuronal Lineal.

Introducción y Objetivos

El grupo debe aplicar las técnicas de la materia Álgebra Lineal Computacional y el módulo ALC construido durante el curso del laboratorio para la resolución de la consigna de este Trabajo Práctico.

Red Neuronal Lineal

Arquitectura Lineal

La red neuronal lineal es el modelo más simple luego del perceptrón. Esta arquitectura consta de una capa de neuronas de entrada $\mathbf{x} \in \mathbb{R}^n$ y una capa neuronas de salida $\mathbf{y} \in \mathbb{R}^m$. Además, cada neurona x_i con $i = 1, \ldots, n$ está conectada con una y_j con $j = 1, \ldots, m$ a través de una sinapsis ponderada por un peso w_{ji}. La operación efectuada por la Red Lineal se corresponde a la suma ponderada mediante la operación siguiente:

$$W\mathbf{x} = \mathbf{y} \tag{1}$$

o bien,

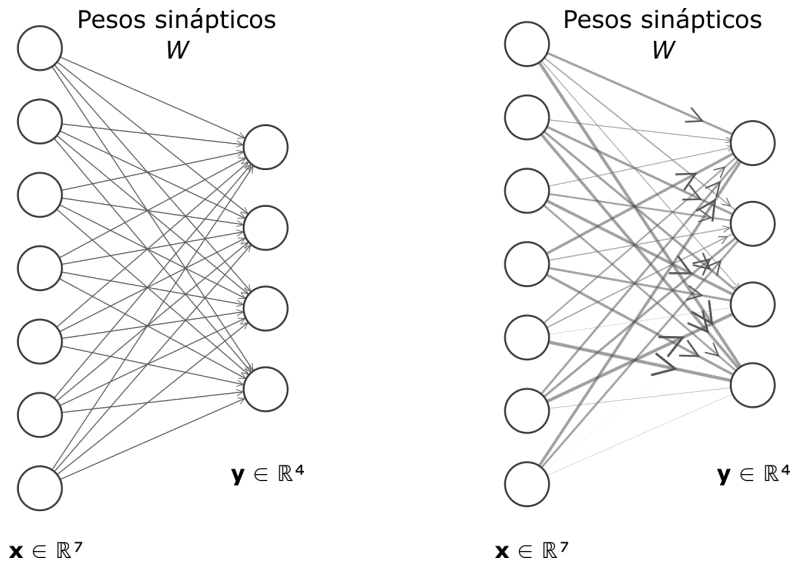
$$y_j = \sum_{i=1}^{n} w_{ji} x_i \quad \forall j = 1, \ldots, m \tag{2}$$

El fin de la red neuronal es ajustar los pesos W de las sinapsis, para que un vector de entrada $\mathbf{x}$ produzca una salida $\mathbf{y}_{target}$ deseada, que llamaremos *target*.

Aprendizaje de la red neuronal

La idea de proyectar la entrada $\mathbf{x}$ mediante los pesos W a una salida esperada $\mathbf{y}$ es muy interesante, pero claramente no evidente. Las w_{ji} de las sinapsis de la ecuación 2 son las responsables que ajustan esta transformación. Si los w_{ji} valiesen cualquier cosa, evidentemente no funcionaría la proyección. Toda la tarea de los sistemas modernos de inteligencia artificial es ajustar estos pesos en arquitecturas considerablemente más complejas, para poder representar cualquier tipo de salida deseada.

En la práctica, los valores de las variables w_{ji} se pueden obtener mediante algoritmos iterativos que van minimizando un error cometido por el sistema en cada ciclo del bucle. Esto se llama habitualmente **entrenamiento**, donde las sinapsis que ayudan a resolver el problema de la proyección se refuerzan aumentando su w_{ji}. Por otro lado, cuando una conexión no es preponderante en disminuir el error de la salida, su fuerza disminuye. La figura 1 muestra cómo las sinapsis se refuerzan en algunas conexiones y otras disminuyen. El algoritmo de entrenamiento por excelencia se denomina *backpropagation*, pero este queda fuera del *scope* de este trabajo práctico.



(a) Arquitectura simple.

(b) Pesos entrenados.

Figura 1: Red Neuronal Lineal.

Problema de clasificación

La clasificación significa poder asignar a una entrada $\mathbf{x}$ una categoría a partir de una lista disponible. Ejemplos abundan en este sentido, como por ejemplo:

- reconocer un rostro en una lista de identidades;
- identificar la voz grabada;
- detectar el sentimiento en un posteo de Twitter;
- identificar la próxima palabra en una oración.

En este caso, las entradas $\mathbf{x}$ pueden ser imágenes, texto, archivos de audio, etc. Por otro lado las salidas pueden ser nombres de personas, palabras o valores categóricos como $\{positivos, negativos\}$. Para este caso de problemas, cada elemento y_j de la salida $\mathbf{y}$ corresponde a la probabilidad de que la entrada pertenezca a una persona o valor categórico del índice j. La asociación se corresponde entonces a buscar la máxima probabilidad en el vector completo $\mathbf{y}$.

Matemáticamente, el problema de clasificación asocia a una entrada $\mathbf{x}$ una clase c perteneciente a un set de clases C, mediante la función f

$$c = \underset{j=1,\ldots,C}{\operatorname{argmax}} f(\mathbf{x}) = \underset{j=1,\ldots,C}{\operatorname{argmax}} y_j$$

Para ello, durante el entrenamiento, la red lineal vincula los p pares:

$$(\mathbf{x}^1, \mathbf{y}^1), \ldots, (\mathbf{x}^\mu, \mathbf{y}^\mu), \ldots, (\mathbf{x}^p, \mathbf{y}^p)$$

Se busca entonces la matriz de pesos W, tal que ante entradas similares a $\mathbf{x}^\mu$ responda con salidas similares a $\mathbf{y}^\mu$. Para ello, la regla de aprendizaje utiliza los patrones de entrada con sus respectivas salidas deseadas para establecer el conjunto óptimo de pesos W.

Si no es posible escoger una matriz de pesos tal que las ecuaciones

$$W\mathbf{x}^\mu = \mathbf{y}^\mu \quad \forall \mu = 1, \ldots, p$$

sean satisfechas en forma **exacta**, interesa entonces encontrar una solución aproximada.

En este sentido, los algoritmos de aprendizaje se deducen a partir de una función que mida el error a la salida de la red, por ejemplo, el error cuadrático medio:

$$E(W) = \frac{1}{p}\sum_{\mu=1}^{p} \|\mathbf{y}^\mu - W\mathbf{x}^\mu\|_2^2 = \frac{1}{p}\sum_{\mu=1}^{p}\sum_{i=1}^{n}(y_i^\mu - W_i x_i^\mu)^2$$

Aprendizaje de la Red Neuronal Lineal

En el presente Trabajo Práctico se va a utilizar un ajuste de los pesos de la matriz W utilizando los métodos vistos en la materia. El método elegido estará apoyado en los Mínimos Cuadrados, y el cálculo de la Pseudo-Inversa.

A partir de la norma de Frobenius $\|M\|_F^2$ de una matriz $M \in \mathbb{R}^{p\times q}$ tal que:

$$\|M\|_F^2 = \|(m_{ij})\|_F^2 = \sum_{i=1}^{p}\sum_{j=1}^{q} m_{ij}^2$$

Luego, la ecuación de $E(W)$ se puede expresar:

$$E(W) = \frac{1}{p}\|Y - WX\|_F^2$$

A partir de esta formulación, una regla de aprendizaje basada en la utilización de la pseudo-inversa de Moore-Penrose puede escribirse como:

$$W = YX^+$$

donde X^+ es la pseudo-inversa de X. Esta elección para W minimiza el error cuadrático medio.

Definición de la pseudo-inversa

La matrix X^+ queda definida como la inversa generalizada, o también conocida como la pseudoinversa o inversa de Moore-Penrose. El concepto de pseudoinversa de Moore-Penrose es útil en el tratamiento de problemas de optimización y de resolución de los mínimos cuadrados. Siendo $A \in \mathbb{R}^{m\times n}$, formalmente podemos establecer que la matriz pseudoinversa $A^+ \in \mathbb{R}^{n\times m}$ está definida como la única matriz que cumple con las cuatro condiciones de Moore-Penrose:

1. $AA^+A = A$,
2. $A^+AA^+ = A^+$,
3. $(AA^+)^T = AA^+$,
4. $(A^+A)^T = A^+A$

Transfer Learning

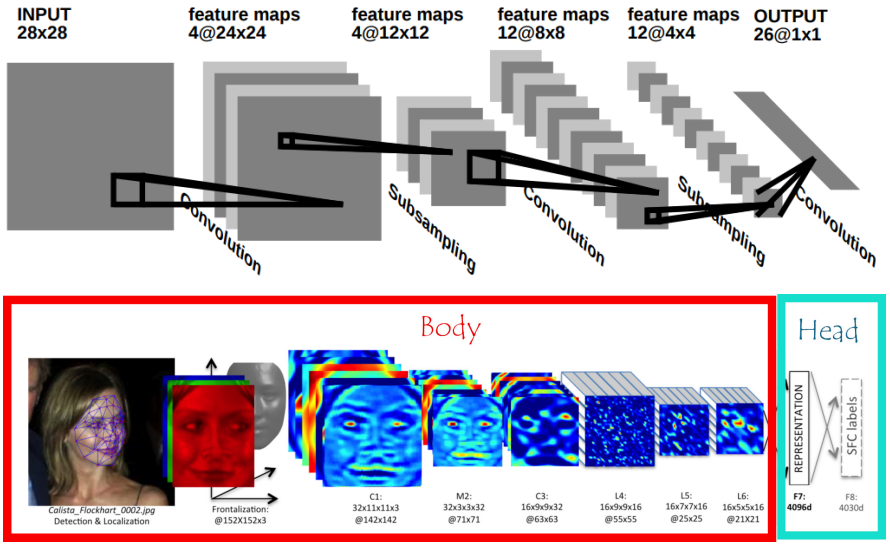
En este TP vamos a utilizar un set de imágenes como entrada al sistema de clasificación. El objetivo es identificar el contenido de cada imagen de este dataset. Dado que en la práctica la conversión del contenido visual de una imagen es una especialidad completa del procesamiento de imágenes y aprendizaje automático, vamos a tomar un atajo en este sentido utilizando el Transfer Learning para 'convertir' una imagen en un vector de características.

El Transfer Learning es una de las técnicas más poderosas y utilizadas en el campo del Deep Learning. En lugar de entrenar una red neuronal desde cero, lo que exigiría una enorme cantidad de datos y recursos computacionales, aprovechamos el 'conocimiento' adquirido por un modelo que ya ha sido entrenado en un dataset a gran escala (como ImageNet).

En este trabajo práctico, utilizaremos un modelo pre-entrenado de última generación, como EfficientNet, para resolver un problema de clasificación de imágenes con un dataset más pequeño. El objetivo es entender cómo adaptar la capa final de clasificación para una nueva tarea, entrenando únicamente una fracción mínima de los parámetros del modelo.

Fundamento Teórico: Anatomía de una Red Convolucional

Una Red Neuronal Convolucional (CNN) moderna, como EfficientNet, puede dividirse conceptualmente en dos partes principales: el módulo convolucional (backbone) y la cabeza de clasificación (head).



El Módulo Convolucional: El Extractor de Características

La primera parte del modelo es una secuencia de capas convolucionales, de pooling y de normalización. Su única función es procesar una imagen de entrada y transformarla en una representación vectorial densa y de alto nivel semántico codificando la información visual. La primera parte de la red, se denomina cuerpo (o body en inglés). Al vector obtenido a la salida del cuerpo lo llamamos embedding.

Este módulo, entrenado sobre millones de imágenes, aprende a detectar características jerárquicas: desde bordes y texturas en las primeras capas hasta formas complejas, objetos y patrones en las capas más profundas. En el caso de EfficientNet, esta arquitectura

está optimizada para obtener la mejor performance con un uso muy eficiente de los recursos computacionales.

Al final de este módulo, justo antes de la capa de clasificación, obtenemos el embedding. Por ejemplo, para una imagen de entrada de 224×224×3 píxeles, el módulo convolucional de una EfficientNetB0 podría generar un embedding $\mathbf{x}$ de dimensión 1536. Este vector ya no contiene píxeles, sino una representación abstracta y rica de los contenidos de la imagen.

La Cabeza de Clasificación: De Features a Predicciones

La segunda parte del modelo de clasificación convolucional es la cabeza de clasificación. Aquí es donde se toma la decisión final a partir de la entrada $\mathbf{x}$ y se estima la salida $\mathbf{y}$. En los modelos pre-entrenados, esta cabeza está diseñada para clasificar las 1000 categorías de ImageNet, nuestro caso, para la tarea planteada en nuestro TP la descartaremos y construiremos una nueva.

Analicemos cómo actúa la Cabeza del modelo. Esta parte se llama Fully Connected (capa Densa) y corresponde al modelo de Red Neuronal Lineal de la figura 1. Esta capa es, en esencia, una simple pero potente transformación lineal representada con la matriz de pesos W.

El proceso es el siguiente:

- El embedding $\mathbf{x}$, un vector de $\mathbb{R}^{1536}$ (para EfficientNetB0), se obtiene como salida del módulo convolucional.
- Se multiplica $\mathbf{x}$ por la matriz W que ajusta una entrada a calcular una probabilidad para 2 clases. Por ende, la matriz W pertenece a $\mathbb{R}^{2\times1536}$.
- El resultado es un vector de salida $\mathbf{y}$ de $\mathbb{R}^2$, conocido como logits y calculado como la ecuación 2.

Dataset de imágenes

El Trabajo Práctico se enfoca en resolver el problema del Transfer Learning aplicado sobre los pesos pre-entrenados del modelo EfficientNet-b3 para que resuelva una base de datos nueva, no vista durante el entrenamiento.

Ejemplos del dataset dogs-and-cats

El dataset $\mathcal{H}$ esta compuesto de fotografías categorizadas en 2 categorías: perros y gatos. El set completo contiene 3.000 imágenes de entrenamiento y 2.000 imágenes de test o validación. Podemos representar el cuerpo del modelo EfficientNet-b3 como una función no lineal b tal que al pasarle como entrada una imagen I_i genera un embedding de salida $\mathbf{x}_i$:

$$b(I_i) = \mathbf{x}_i, \forall\ I_i \in \mathcal{H}, \mathbf{x}_i \in \mathbb{R}^{1536}$$

Cada imagen I_i tiene asociada la clase correspondiente al animal, generando una tupla (I_i, t) con $t \in \{0, 1\}$. Por simplicidad, vamos a agregar la clase del animal a la imagen y al embedding como un supra-índice tal que quede:

$$b(I_i^t) = \mathbf{x}_i^t, \forall\ I_i^t \in \mathcal{H}, \mathbf{x}_i^t \in \mathbb{R}^{1536},\ t \in \{0, 1\}$$

La tarea de transfer learning, o finetunning, consiste en reemplazar la capa head del fully connected W por una nueva matriz W_{ft} tal que:

$$W_{ft}\ \mathbf{x}_i^t = \mathbf{e}_t$$

donde $\mathbf{e}_t$ es un vector canónico con todos valores igual a cero, excepto en el índice t que tiene valor 1.

Resolución de la Capa Densa de salida

Presentamos a continuación tres opciones de algoritmos para calcular los pesos de la última capa densa en la matriz W. Los algoritmos se basan en le cálculo de la Pseudo-Inversa de la matriz de embeddings, pero utilizando diferentes descomposiciones matriciales.

Algorithm 1 Fully Connected Lineal con Cholesky en Rango Completo

Require: Dada $X \in \mathbb{R}^{n \times p}$ y $Y \in \mathbb{R}^{m \times p}$

Ensure: Solución W del problema $\text{mín}_W \|Y - WX\|_2$

1: a- Si $rango(X) = p$ $n > p$, entonces $X^+ = (X^T X)^{-1} X^T$.

2: b- Si $rango(X) = n$ $n < p$, entonces $X^+ = X^T (X X^T)^{-1}$.

3: c- Si $rango(X) = n$ y $n = p$, entonces $X^+ = X^{-1}$

4: Calcular $W = Y X^+$

- para (a), resolver el sistema $(X^T X)U = X^T$ aplicando Cholesky a $(X^T X)$. Luego despejar W de $W = YU$,
- para (b), resolver el sistema $V(X X^T) = X^T$ aplicando Cholesky a $(X X^T)$. Luego despejar W de $W = YV$,
- para (c), despejar W de $WX = Y$,

Algorithm 2 Fully Connected Lineal con Descomposición en Valores Singulares

Require: Dada $X \in \mathbb{R}^{n \times p}$ con $n < p$, $rango(X) = n$ (rango completo) y $Y \in \mathbb{R}^{m \times p}$

Ensure: Solución del problema $\text{mín}_W \|Y - WX\|_2$

1: Calcular la Descomposición en valores singulares de X tal que $X = U \Sigma V^T$ donde $U \in \mathbb{R}^{n \times n}$ y $V \in \mathbb{R}^{p \times p}$ son ortogonales y

$$\Sigma = \begin{bmatrix} \Sigma_1 & 0 \end{bmatrix} \; \Sigma_1 = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ & & \ddots & \\ 0 & 0 & \dots & \sigma_n \end{bmatrix} \in \mathbb{R}^{n \times n}$$

2: Calcular $X^+ = V \Sigma^+ U^T$ donde

$$\Sigma^+ = \begin{bmatrix} \Sigma_1^{-1} \\ 0 \end{bmatrix}$$

3: Calcular la solución dada por $W = Y V \Sigma^+ U^T$ para ello

- Particionar las matrices U y V en columnas: $U = [U_1 \; U_2]$ y $V = [V_1 \; V_2]$.
- Calcular el producto $V\Sigma^+ = \begin{bmatrix} V_1 & V_2 \end{bmatrix} \begin{bmatrix} \Sigma_1^{-1} \\ 0 \end{bmatrix} = \begin{bmatrix} V_1 \Sigma_1^{-1} + 0 \end{bmatrix}$
- Finalmente $V\Sigma^+ U^T = \begin{bmatrix} V_1 \Sigma_1^{-1} \end{bmatrix} \begin{bmatrix} U_1^T \end{bmatrix} = \begin{bmatrix} V_1 \Sigma_1^{-1} U_1^T \end{bmatrix}$ para encontrar W

Enunciado

El TP busca que se utilicen las funciones desarrolladas durante el curso del laboratorio y volcadas en el módulo `alc.py`. Estas funciones están pensadas para reemplazar la librería

Algorithm 3 Fully Connected Lineal con Descomposición QR

Require: Dada $X \in \mathbb{R}^{n \times p}$ y $Y \in \mathbb{R}^{m \times p}$

Ensure: Solución del problema $\text{mín}_W \|Y - WX\|_2$

1: Vamos a resolver la solución a derecha $X^+ = X^T(XX^T)^{-1}$.
2: Dado que $X \in \mathbb{R}^{n \times p}$ con $n < p$, $rango(X) = n$ (rango completo), calculamos QR sobre X^T: $X^T = QR$ usando Householder (o Gram-Schmidt)
3: Sustituir en X^+; $X^+ = (QR)\left((R^TQ^T)(QR)\right)^{-1} = Q\left(R^T\right)^{-1}$
4: Multiplicamos ambos lados a la derecha por X^T: $X^+R^T = Q$.
5: Llamamos $V = X^+$ y resolvemos para V en $VR^T = Q$
6: Obtenemos $W = YV$

`numpy` de Python, de la cual no está permitido utilizar las funciones ya marcadas en los laboratorios.

Los materiales utilizados para el TP son los siguientes:

- `dataset-cats-dogs.zip`: Es el dataset de los embeddings ya calculados del dataset de entrenamiento (train) y validación (val) utilizando el modelo EfficientNet.

1. **Lectura de datos**

   a. Implementar la función `cargarDataset(carpeta)` que devuelve las siguientes matrices X_t, Y_t, X_v, Y_v. Las matrices (X_t, Y_t) corresponden a los ejemplos de entrenamiento conteniendo los embeddings de los gatos y perros juntos. Cada columna de la matriz corresponde a un embedding. La matriz Y_t debe generarse a partir de la lectura de los archivos de entrenamiento. Cada columna de Y_t tiene 2 elementos valiendo 0 o 1 dependiendo de la clase a la que pertenece el embedding. Por ejemplo un y_i de un gato debería ser $y_i = [1, 0]^T$, y otro de un perro: $y_j = [0, 1]^T$.

2. **Ecuaciones Normales**

   a. Implementar el algoritmo 1 que resuelve el cálculo de los pesos W utilizando las ecuaciones normales para la resolución de la pseudo-inversa. La función de Cholesky debe ser la implementación propia del grupo. La función se denomina `pinvEcuacionesNormales(L, Y)`. La función recibe la matriz X de los embeddings de entrenamiento, L la matriz de Cholesky, y Y la matriz de targets de entrenamiento. La función devuelve W.

3. **Descomposición en Valores Singulares**

   a. Implementar el algoritmo 2 que obtiene los pesos W utilizando la Descomposición en Valores Singulares para la resolución de la pseudo-inversa. La función para calcular U, S, V debe ser la implementación propia del grupo. La función se denomina `pinvSVD(U, S, V, Y)`. La función recibe la matriz X de los embeddings de entrenamiento, las matrices U, S, V de la descomposición SVD, y Y la matriz de targets de entrenamiento. La función devuelve W.

4. **Descomposición QR** Implementar el algoritmo 3 que calcula los pesos W a través de la descomposición QR.

a. HouseHolder: la función se denomina `pinvHouseHolder(Q, R, Y)`. La función recibe la matriz X de los embeddings de entrenamiento, las matrices Q, R de la descomposición QR utilizando HouseHolder, y Y la matriz de targets de entrenamiento. La función devuelve W.

b. Gram-Schmidt: la función se denomina `pinvGramSchmidt(Q, R, Y)`. La función recibe la matriz X de los embeddings de entrenamiento, las matrices Q, R de la descomposición QR utilizando GramSmidth clásico, y Y la matriz de targets de entrenamiento. La función devuelve W.

5. **Pseudo-Inversa de Moore-Penrose**

   a. Implementar la función `esPseudoInverda(X, pX, tol=1e-8)`, que reciba dos matrices y devuelva `True` si verifican las condiciones de Moore-Penrose. En caso contrario devolver `False`.

6. **Evaluación y Benchmarking**:

   a. Para cada método de resolución de la W de los items anteriores, generar una matriz de confusión evaluando a partir de los pares de embeddings de validación o testing (X_v, Y_v).

   b. Presentar una tabla comparativa de los resultados de cada metodología donde en las columnas se debe mostrar la performance en clasificación, y el número de operaciones.

7. **Síntesis final.** Habiendo obtenido las tablas de resultados, analizar la performance de cada metodología y las posibles diferencias encontradas. El texto debe ser de alrededor de 400 palabras.

Template-alumnos, Defensa Oral y Entrega

En el campus, en los recursos para el TP van a encontrar los siguientes archivos:

- Archivo `tp2c2025.pdf`: este archivo de enunciado del TP.
- Archivo `template-alumnos.zip`: archivo de recursos con el siguiente contenido:
  - Archivo `dataset.zip`: Este archivo contiene 2 carpetas llamadas *train* y *val*. Cada una de estas carpetas tiene a su vez dos carpetas denominadas **cats** y **dogs**. Dentro de estas carpetas encontrarán ya calculados los embeddings de efficientNet como las matrices $X \in \mathbb{R}^{1536\times1000}$, para train y $X \in \mathbb{R}^{1536\times500}$ validación.
  - Folder `ejemplo`: contiene un script que muestra como fueron obtenidos los embeddings de las imágenes usando el modelo efficientNet pre-entrenado para la clasificación del dataset IMAGENET.

La entrega se realizará a través del campus virtual de la materia además de una defensa oral grupal del trajo. Las fechas y formatos son los siguientes:

- Fecha de entrega: hasta el Miércoles **19 de Noviembre** a las 23:59 hs.

- Fecha de defensa oral: Martes **2 de Diciembre**.

- Formato items 1-5: archivo biblioteca con todas las funciones pedidas en el TP y del labo. Nombre del archivo: alc.py.

- Formato item 6-7: Jupyter Notebook.

Prestar especial atención a las siguientes indicaciones:

- El TP se realizará en grupos de cuatro personas. Indicar a los docentes si el grupo declarado en el 'Foro de Grupos de TP' no ha sido aún creado. **Importante:** es indispensable realizar la inscripción previa del grupo para poder hacer el envío a través del campus. Los grupos o personas no inscriptas en grupos no estarán habilitadas en el formulario de carga del TP. No se aceptarán envíos por email.

- Leer el enunciado completo antes de comenzar a generar código y sacarse todas las dudas de cada ítem antes de implementar. Para obtener un código más legible y organizado, pensar de antemano qué funciones deberán implementarse y cuáles podrían reutilizarse.

- El código debe estar correctamente comentado. Cada función definida debe contener un encabezado donde se explique los parámetros que recibe y qué se espera que retorne. Además las secciones de código dentro de la función deben estar debidamente comentados. Los nombre de las variables deben ser explicativos.

- Las conclusiones y razonamientos que respondan los ejercicios, o cualquier experimentación agregada, debe estar debidamente explicada en bloques de texto de las notebooks (markdown cells), separado de los bloques de código. Aprovechen a utilizar código LaTeX si necesitan incluir fórmulas.