2024/2025 1Q

Lista de Problemas 3

APA

Javier Béjar

Departament de Ciències de la Computació

Grau en Enginyeria Informàtica - UPC

UPC FIB Facultat d'Informàtica de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

DEPARTAMENT DE CIÉNCIES DE LA COMPUTACIÓ

FACULTAT D'INFORMÁTICA DE BARCELONA

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Primera edición, septiembre 2021

Esta edición, Septiembre 2024

Instrucciones:

Para la entrega de grupo debéis elegir un problema del capítulo de problemas de grupo.

Para la entrega individual debéis elegir un problema del capítulo de problemas individuales. **Cada miembro del grupo debe elegir un problema diferente**.

Debéis hacer la entrega subiendo la solución al racó.

Evaluación:

La nota de esta entrega se calculará como 1/3 de la nota del problema de grupo más 2/3 de la nota del problema individual.

Al realizar el informe correspondiente a los problemas explicad los resultados y las respuestas a las preguntas de la manera que os parezca necesaria. Se valorará más que uséis gráficas u otros elementos para ser más ilustrativos.

Podéis entregar los resultados como un notebook (Colab/Jupyter). Alternativamente, podéis hacer un documento explicando los resultados como un PDF y un archivo python con el código

También, si queréis, podéis poner las respuestas a las preguntas en el notebook, este os permite insertar texto en markdown y en latex.

Objetivos de aprendizaje:

1. Conocer las arquitecturas de perceptrón multicapa y redes convolucionales
2. Conocer las funciones de error según el tipo de tarea de un problema
3. Experimentar con diferentes configuraciones para una red y saberlas ajustar para diferentes tipos de problemas

CAPÍTULO 1

Problemas de Grupo

Al resolver el problema explicad bien los que hacéis, no hacer ningún comentario o hacer comentarios superficiales tendrán una nota más baja.

Tenéis que mostrar que habéis entendido los métodos que estáis aplicando, así que un corta y pega de problemas similares no es suficiente.

Estos problemas se han de resolver utilizando la librería PyTorch. Esto significa que deberéis hacer la experimentación usando Google Colab para poder hacerla en un tiempo razonable.

Aunque hagáis los experimentos en Colab necesitaréis cierto tiempo para obtener los resultados, así que no esperéis hasta el último momento para hacerlos.

1. **¿Qué número es este?**

   Parece haber una obsesión con conjuntos de datos que representan dígitos escritos a mano. Esto tiene una explicación sencilla. Uno de los primeros retos del OCR fue el reconocimiento de códigos postales en cartas. El clasificar cartas a mano consume demasiado tiempo y no es completamente fiable (además de ser muy aburrido), por lo que tener un sistema eficiente para clasificar el correo era un problema interesante[1]. Uno de los conjuntos de datos más conocido es MNIST digits. Vamos a utilizar una parte de este conjunto de datos, en concreto un subconjunto de los dígitos 3, 5, 6, 8 y 9, para experimentar con diferentes redes neuronales. Podéis obtenerlos mediante la función `load_MNIST` de la librería `apafib`. Esta función retornará cuatro matrices de datos, el conjunto de entrenamiento, el conjunto de test y las etiquetas de los dos. Resolved los siguientes apartados ilustrando los resultados de la manera que os parezca más adecuada.

   Para resolver los siguientes apartados tenéis un notebook con una serie de funciones que os servirán para resolverlos.

   a) Los datos corresponden a imágenes de 28×28 píxeles en niveles de gris. Estos ya están convertidos a vectores y normalizados a la escala [0,1]. Aplica PCA y t-SNE a los datos

[1] Ahora ya nadie escribe cartas, pero siempre habrá texto escrito a mano que reconocer en otras aplicaciones.

de entrenamiento y represéntalos en 2D. ¿Se puede ver separabilidad entre las clases? Comenta el resultado.

b) Comenzaremos por obtener un valor de acierto base. Asumiendo que los píxeles son independientes y se distribuyen de manera Multinomial (solo hay 16 valores distintos en los datos). Ajustad un modelo Naïve Bayes Multinomial. Evaluad la calidad del modelo y comentad los resultados.

c) Ahora aplicaremos un MLP a estos datos. En el código del notebook tenéis una función que permite generar modelos MLP variando el número de capas y la función de activación. La red se define como una clase python donde en el método `__init__` definimos los elementos de la red y en el `forward` cómo se hace la propagación hacia adelante en la red (la propagación hacia atrás se calcula automáticamente).

Para pasarle los datos a un modelo de `torch` hay que definir una clase del tipo `Dataset`, tenéis una definida en el notebook a la que hay que pasarle la matriz de datos y el vector con las etiquetas. Esta clase se ha de pasar a un objeto `Dataloader` que se encarga de organizar el uso de los datos durante el entrenamiento, entre otras cosas de partir los datos en grupos (`batch_size`), podéis ver un ejemplo en el notebook.

Para entrenar los modelos usaremos la función `train_loop` que tenéis en el notebook. Esta función recibe el modelo, el optimizador a usar, los datos de entrenamiento y validación y el número de iteraciones que hará el entrenamiento. Para el optimizador usaremos `adam`, podéis ver en el notebook como generar el objeto del optimizador.

Entrenad diferentes redes de 2 y 3 capas con diferentes tamaños usando como función de activación `ReLU` y `Sigmoid`. Para los tamaños podéis elegir el tamaño de la primera capa oculta y después ir reduciéndolo a la mitad para las capas sucesivas. Para crear la red solo tenéis que crear un objeto de la clase `MLP` pasándole los parámetros adecuados. Al generar el objeto tendréis que subir el modelo a la GPU usando el método `modelo.to('cuda')` (si no lo hacéis torch se quejará de que los datos y el modelo no están en el mismo dispositivo). Veréis que la función de entrenamiento retorna la historia de la función de pérdida para el entrenamiento y validación. Usad esta información para comprobar la sobre especialización de la red y el número de épocas que realmente se necesita para converger.

Evaluad los resultados. Para ello podéis usar la función `test_model` que tenéis en el notebook. Se le ha de pasar el modelo entrenado y un conjunto de datos y retornará las etiquetas de las predicciones y las reales, a partir de ahí podéis usar las funciones de `scikit learn` para obtener las medidas de evaluación.

d) Las redes MLP tienen bastantes parámetros y son menos eficientes que las redes convolucionales en tareas de imágenes. Tenéis en el notebook una clase que permite definir redes convolucionales para los datos del problema. Entrenad redes de dos capas que tengan diferentes números de kernels convolucionales, para ello, elegid el número de kernels para la primera capa y dobladlo para la segunda. Podéis evaluar la sobre especialización del modelo y evaluarlo de la misma manera que en el apartado anterior. Comparad los resultados.

e) Podemos visualizar el efecto tiene la función aprendida por las redes obteniendo el resultado antes de la capa de salida (la capa lineal final). Añadid un método a las clases que definen las redes que apliquen a la entrada las capas de la red excepto esa última capa. Obtened el resultado de esta transformación a partir de las mejores redes entrenadas para el MLP con ReLU y sigmoide y la convolucional. Aplicad un PCA a ese resultado y visualizadlo. Comentad los resultados.

2. **Inspeccionando el tráfico**

CIFAR-10[2] es uno de los muchos conjuntos de imágenes que existen para poder hacer experi-

[2]CIFAR es el acrónimo del Canadian Institute For Advanced Research.

mentos. Este conjunto tiene imágenes de 32×32 de 10 clases diferentes. Dos de esas clases son coche y camión. Uno pensaría que debería ser fácil el poder distinguir ejemplos de estas dos clases, el detalle es que muchas veces son imágenes en el que solo aparece parcialmente el objeto, o este puede ocupar una parte relativamente pequeña de la imagen. El objetivo de este problema es experimentar con diferentes redes MLP y convolucionales para ver cuál es el nivel de acierto al que se puede llegar.

Podéis obtenerlos mediante la función `load_vehiculos` de la librería `apafib`. Esta función retornará dos matrices, una con las imágenes ($3 \times 32 \times 32$) y otra con las etiquetas. Resolved los siguientes apartados ilustrando los resultados de la manera que os parezca más adecuada.

Para resolver los siguientes apartados tenéis un notebook con una serie de funciones que os servirán para resolverlos.

a) Los datos corresponden a imágenes de 32×32 píxeles en colores RGB (cada color es una matriz 2D). Están ya normalizados a la escala [0,1]. Aplicad PCA y t-SNE a los datos de entrenamiento y representadlos en 2D. ¿Se puede ver separabilidad entre las clases? Comentad el resultado.

b) Empezaremos aplicando un MLP a estos datos. En el código del notebook tenéis una función que permite generar modelos MLP variando el número de capas y la función de activación. La red se define como una clase python donde en el método `__init__` definimos los elementos de la red y en el `forward` cómo se hace la propagación hacia adelante en la red (la propagación hacia atrás se calcula automáticamente).

Para poder entrenar el modelo primero partiremos el conjunto de datos en tres, el conjunto de entrenamiento propiamente dicho, un conjunto de validación y un conjunto de test (70 %/15 %/15 %), ya que no haremos validación cruzada. Para pasarle los datos a un modelo de `torch` hay que definir una clase del tipo `Dataset`, tenéis una definida en el notebook a la que hay que pasarle la matriz de datos y el vector con las etiquetas. Esta clase se ha de pasar a un objeto `Dataloader` que se encarga de organizar el uso de los datos durante el entrenamiento, entre otras cosas de partir los datos en grupos (`batch_size`), podéis ver un ejemplo en el notebook.

Para entrenar los modelos usaremos la función `train_loop` que tenéis en el notebook. Esta función recibe el modelo, el optimizador a usar, los datos de entrenamiento y validación, la paciencia para la terminación temprana y el número de iteraciones que hará el entrenamiento. Para el optimizador usaremos `adam`, podéis ver en el notebook como generar el objeto del optimizador.

Entrenad diferentes redes de 2 y 3 capas con diferentes tamaños usando como función de activación `Sigmoid`. Para los tamaños podéis elegir el tamaño de la primera capa oculta y después ir reduciéndolo a la mitad para las capas sucesivas. Para crear la red solo tenéis que crear un objeto de la clase `MLP` pasándole los parámetros adecuados. Al generar el objeto tendréis que subir el modelo a la GPU usando el método `modelo.to('cuda')` (si no lo hacéis torch se quejará de que los datos y el modelo no están en el mismo dispositivo). Veréis que la función de entrenamiento retorna la historia de la función de pérdida para el entrenamiento y validación. Usad esta información para comprobar la sobre especialización de la red y el número de épocas que realmente se realizan antes de parar. Podéis dejar la paciencia en su valor por defecto.

Evaluad los resultados. Para ello podéis usar la función `test_model` que tenéis en el notebook. Se le ha de pasar el modelo entrenado y un conjunto de datos y retornará las etiquetas de las predicciones y las reales, a partir de ahí podéis usar las funciones de `scikit learn` para obtener las medidas de evaluación.

c) Las redes MLP tienen bastantes parámetros y son menos eficientes que las redes convolucionales en tareas de imágenes. Tenéis en el notebook una clase que permite definir redes convolucionales para los datos del problema. Entrenad redes de dos capas que tengan diferentes números de kernels convolucionales, para ello, elegid el número de kernels para la primera capa y dobladlo para la segunda. Podéis evaluar la sobre especialización del modelo y evaluarlo de la misma manera que en el apartado anterior. Comparad los resultados.

d) La red convolucional transforma el resultado de las dos capas convolucionales usando una capa lineal final que da la clasificación. Igual introduciendo un MLP antes de esa capa lineal obtenemos una mejor clasificación. Modificad la clase de la red convolucional introduciendo antes de la capa lineal otra capa lineal que transforme la entrada a un tamaño intermedio y después aplique una función de activación `ReLU`. Ahora la capa lineal final tendrá que transformar de ese tamaño intermedio al tamaño de la salida. Añadid el tamaño de la nueva capa como un parámetro de la clase. Experimentad con diferentes tamaños intermedios con la red convolucional que mejor resultados os dio en el apartado anterior. Podéis evaluar la sobre especialización del modelo y evaluarlo de la misma manera que en apartados anteriores. Comparad los resultados.

e) Habréis visto que las implementaciones de las redes tienen una capa llamada `Dropout` que esta antes de la salida lineal. Esta capa hace cero de manera aleatoria las activaciones que le llegan. Esto evita que la red dependa específicamente de alguna de las entradas que le llegan a la capa de salida y actúa como una regularización. Esta capa tiene puesto un valor de `0.0` por defecto. Experimentad con la red que os haya dado el mejor resultado usando valores de dropout `0.1`, `0.25` y `0.75`. Usad valores de paciencia de 5 y 10. Podéis evaluar la sobre especialización del modelo y evaluarlo de la misma manera que en apartados anteriores. Comparad los resultados.

3. **El aire que respiramos**

Barcelona es una ciudad que se preocupa por su aire, para ello tiene estaciones de medición en diferentes puntos de la ciudad. La estación de Palau Reial permite obtener datos horarios de concentraciones de diferentes contaminantes, entre ellos NO2, O3, SO2, PM2.5 y PM10. Utilizaremos datos que corresponden a los años 2022 y 2023 para obtener un modelo con redes neuronales que permita hacer predicción a una hora del NO2. Podéis obtener los datos mediante la función `load_BCN_air` de la librería `apafib`. Esta función retornará una matriz de datos con las series temporales para cada variable. Resolved los siguientes apartados ilustrando los resultados de la manera que os parezca más adecuada.

Para resolver los siguientes apartados tenéis un notebook con una serie de funciones que os servirán para resolverlos.

a) Primero haremos una limpieza de los datos. Veréis que existen datos perdidos en todas las variables. Dado que son series temporales que tienen una periodicidad horaria, los valores faltantes no deberían distar mucho de los más cercanos. Una opción para imputarlos es usar el método `interpolate` de los data frames de Pandas. Podemos usar interpolación lineal, de manera que cada dato faltante sea el valor de una interpolación lineal entre los extremos.

Para generar los datos necesitaremos ventanas temporales de cierta longitud. `Pandas` permite generar una copia de una tabla de datos desplazada una serie de instantes temporales usando el método `shift`. Intentaremos hacer la predicción a partir de los datos de las últimas seis horas, así que tendréis que aplicar la operación para obtener seis matrices desplazadas y concatenarlas con los datos originales Eliminad todos los valores perdidos que ha generado esta transformación. Partid el conjunto de datos en entrenamiento, validación

y test (70 %/15 %/15 %). Descartad las ventanas entre las particiones que tienen instantes compartidos para asegurar que las particiones sean independientes. Normalizad las particiones para que estén en el rango 0-1 de manera adecuada.

Para poder tratar los datos con las redes neuronales para cada partición de datos tendréis que extraer la matriz de datos, separar las columnas del último instante temporal del resto, obtener solo la primera columna del último instante temporal (la variable NO2) y transformar la matriz de datos que usaremos para predecir de manera que tenga la forma `[ejemplos, variables, tiempo]` (tendréis que cambiar las dimensiones e intercambiar los dos últimos ejes entre si).

Ahora ya tenéis tres conjuntos de datos que podéis usar para entrenar modelos (entrenamiento, validación y prueba).

b) Tenemos un problema de regresión, podemos usar un MLP con varias capas ocultas que tenga una salida lineal para hacer la predicción y ajustadlo usando mínimos cuadráticos (mean squared error). En el código del notebook tenéis una función que permite generar modelos MLP variando el número de capas y la función de activación. La red se define como una clase python donde en el método `__init__` definimos los elementos de la red y en el `forward` cómo se hace la propagación hacia adelante en la red (la propagación hacia atrás se calcula automáticamente).

Para pasarle los datos a un modelo de `torch` hay que definir una clase del tipo `Dataset`, tenéis una definida en el notebook a la que hay que pasarle la matriz de datos y los valores a predecir. Esta clase se ha de pasar a un objeto `Dataloader` que se encarga de organizar el uso de los datos durante el entrenamiento, entre otras cosas de partir los datos en grupos (`batch_size`), podéis ver un ejemplo en el notebook.

Para entrenar los modelos usaremos la función `train_loop` que tenéis en el notebook. Esta función recibe el modelo, el optimizador a usar, los datos de entrenamiento y validación, la paciencia para la terminación temprana y el número de iteraciones que hará el entrenamiento. Para el optimizador usaremos `adam`, podéis ver en el notebook como generar el objeto del optimizador.

Entrenad diferentes redes de 3 capas con diferentes tamaños usando como función de activación `Sigmoid` y `ReLU`. Para los tamaños podéis elegir el tamaño de la primera capa oculta y después ir reduciéndolo a la mitad para las capas sucesivas. Para crear la red solo tenéis que crear un objeto de la clase `MLP` pasándole los parámetros adecuados. Al generar el objeto tendréis que subir el modelo a la GPU usando el método `modelo.to('cuda')` (si no lo hacéis torch se quejará de que los datos y el modelo no están en el mismo dispositivo). Veréis que la función de entrenamiento retorna la historia de la función de pérdida para el entrenamiento y validación. Usad esta información para comprobar la sobre especialización de la red y el número de épocas que realmente se realizan antes de parar. Podéis dejar la paciencia en su valor por defecto.

Evaluad los resultados. Para ello podéis usar la función `test_model` que tenéis en el notebook. Se le ha de pasar el modelo entrenado y un conjunto de datos y retornará las etiquetas de las predicciones y las reales, a partir de ahí podéis usar las funciones de `scikit learn` para obtener las medidas de evaluación.

c) Las redes convolucionales también se pueden usar para predecir series temporales. Usaremos redes que usan convoluciones en una sola dimensión, de manera que las variables son los canales y la secuencia los valores. Tenéis en el notebook una clase que permite definir redes convolucionales para los datos del problema. Entrenad redes de dos capas que tengan kernels de tamaños 3 y 5 con diferentes números de kernels convolucionales, para ello, elegid el número de kernels para la primera capa y dobladlo para la segunda. Podéis evaluar la sobre especialización del modelo y evaluarlo de la misma manera que en

el apartado anterior. Comparad los resultados.

d) A veces nos interesa predecir a más de una hora vista, pongamos que nos interesa predecir las próximas tres horas a partir de las seis anteriores. Modificad adecuadamente la clase que encapsula los datos (veréis que añade artificialmente una dimensión para predecir un vector de una columna). Modificad la clase que implementa el MLP para que pueda hacer tres predicciones en lugar de una. Entrenad redes de tres capas con diferentes tamaños usando la función de activación `ReLU`. Evaluad adecuadamente los resultados.

CAPÍTULO 2

Problemas Individuales

Al resolver el problema explicad bien los que hacéis, no hacer ningún comentario o hacer comentarios superficiales tendrán una nota más baja.

Tenéis que mostrar que habéis entendido los métodos que estáis aplicando, así que un corta y pega de problemas similares no es suficiente.

Para obtener los datos para algunos de estos problemas necesitaréis instalaros la última versión de la librería `apafib`. La podéis instalar localmente haciendo:

```
pip install -user -upgrade apafib
```

Para usar las funciones de carga de datos solo tenéis que añadir su importación desde la librería, en vuestro script o notebook, por ejemplo

```
from apafib import load_BCN_Francia
```

La función os retornara un `DataFrame` de Pandas o arrays de numpy con los datos y la variable a predecir, cada problema os indicará que es lo que obtendréis.

Aunque hagáis los experimentos en Colab necesitaréis cierto tiempo para obtener los resultados, así que no esperéis hasta el último momento para hacerlos.

1. **¿Que trae a los Franceses a Barcelona?**

   Francia es un país vecino, y obviamente esa vecindad hace que muchos vengan a visitar Barcelona, pero ¿cuántos vendrán?. El ayuntamiento de Barcelona recolecta diversos datos sobre la ciudad en su portal de datos abiertos[1]. Vamos a trabajar con un extracto de esos datos para los años 2021-2023 con un subconjunto de variables que hemos elegido según nuestro criterio *experto* que incluyen alimentos, datos meteorológicos y llegadas al aeropuerto. El objetivo es aproximar el número de visitantes diarios de ciudadanos Franceses a Barcelona.

[1] https://portaldades.ajuntament.barcelona.cat/

Puedes obtener estos datos mediante la función `load_BCN_Francia` de la librería `apafib`. Resuelve los siguientes apartados ilustrando los resultados de la manera que te parezca más adecuada.

a) Divide el conjunto de datos en entrenamiento y test (80 %/20 %). Haz una exploración mínima del conjunto de datos de entrenamiento observando las relaciones entre las variables, especialmente con la variable objetivo. Describe las cosas que hayas visto que te parezcan interesantes. A veces tener información basada en la fecha puede ayudar en la predicción. El índice de los datos es la fecha, obtén a partir de ella el día, el día de la semana y el mes y añádelos a la matriz de datos. Normaliza las variables adecuadamente para poder ajustar un modelo de regresión tanto para el conjunto de entrenamiento como para el de test.

   El valor a predecir tiene una magnitud bastante grande (decenas de miles). Esto hace difícil a veces el obtener una buena predicción con algunos modelos, ya que los pesos necesarios para obtener la salida son muy grandes. Una transformación común es transformar la variable respuesta aplicando el logaritmo.

b) Aplica Análisis de Componentes Principales (PCA) al conjunto de entrenamiento y visualízalo en 2D representando la variable objetivo. ¿Crees que puede haber una relación entre las variables del conjunto de datos y la variable objetivo?

c) Para tener un valor base primero ajusta una regresión Ridge a los datos. ¿Te parece suficientemente bueno el resultado? Representa los valores de la variable objetivo para el conjunto de test contra las predicciones y calcula los residuos de la predicción.

d) Las redes neuronales nos permiten obtener combinaciones no lineales de los datos que podrían capturar las interacciones entre las variables. Ajusta ahora un MLP explorando sus hiperparámetros adecuadamente. Usa la exploración bayesiana para el ajuste para poder hacer la exploración más eficientemente[2]. Compara los resultados con el modelo anterior usando el MSE.

e) La red neuronal no puede darnos la importancia que tiene cada atributo en la predicción, pero podemos intentar reducir el número de variables que son necesarias para saber si alguna no es relevante. Un algoritmo eficiente de reducción de atributos es hacer una búsqueda avariciosa añadiendo (o eliminado) atributos evaluando el modelo hasta obtener un subconjunto que obtenga el mejor resultado. Tienes una implementación en `scikit-learn` que utiliza validación cruzada para la evaluación (`SequentialFeatureSelector`). Para aplicarlo hay que pasarle un modelo ya ajustado, puedes hacer la búsqueda hacia adelante. Usaremos el modelo MLP del apartado anterior. Tendrás que usar también el MSE como evaluación, Obtén con este método un conjunto de atributos reducido y vuelve a ajustar los hiperparámetros del MLP con estos datos. Compara los resultados.

2. **Sonríe**

   En los inicios de las cámaras digitales de fotos[3] a un fabricante se le ocurrió que sería una buena idea el tener una cámara que detectara cuando se estaba sonriendo (podéis buscar la patente). Lo que entonces era un proceso más o menos algorítmico ahora se puede obtener una mejor precisión con una red neuronal. Vamos a utilizar un pequeño conjunto de imágenes (pequeñas) para entrenar un modelo que sea capaz de hacer esta detección.

   Puedes obtener estos datos mediante la función `load_smile` de la librería `apafib`. Resuelve los siguientes apartados ilustrando los resultados de la manera que te parezca más adecuada.

[2] Fíjate que tendrás que usar el MSE como función para explorar los hiperparámetros, puedes ver como se hace en el notebook de laboratorio correspondiente

[3] Los teléfonos móviles acabaron con ellas.

Para resolver los siguientes apartados tienes un notebook con una serie de funciones que te servirán para resolverlos.

a) Los datos corresponden a imágenes de 32×32 píxeles en colores RGB (cada color es una matriz 2D). Los datos ya estan normalizados a la escala [0,1]. Para poder entrenar el modelo primero partiremos el conjunto de datos en tres, el conjunto de entrenamiento propiamente dicho, un conjunto de validación y un conjunto de test (70 %/15 %/15 %), ya que no haremos validación cruzada. Aplica PCA y t-SNE a los datos de entrenamiento y represéntalos en 2D. ¿Se puede ver separabilidad entre las clases? Comenta el resultado.

b) Usaremos una red convolucional para entrenar el clasificador. En el código del notebook tienes una función que permite generar modelos de red convolucional variando diferentes parámetros. La red se define como una clase python donde en el método `__init__` definimos los elementos de la red y en el `forward` como se hace la propagación hacia adelante en la red (la propagación hacia atrás se calcula automáticamente).

Para pasarle los datos a un modelo de `torch` hay que definir una clase del tipo `Dataset`, tenéis una definida en el notebook a la que hay que pasarle la matriz de datos y el vector con las etiquetas. Esta clase se ha de pasar a un objeto `Dataloader` que se encarga de organizar el uso de los datos durante el entrenamiento, entre otras cosas de partir los datos en grupos (`batch_size`), podéis ver un ejemplo en el notebook.

Para entrenar los modelos usaremos la función `train_loop` que tenéis en el notebook. Esta función recibe el modelo, el optimizador a usar, los datos de entrenamiento y validación, la paciencia para la terminación temprana y el número de iteraciones que hará el entrenamiento. Para el optimizador usaremos `adam`, podéis ver en el notebook como generar el objeto del optimizador.

Empezaremos explorando el número y el tamaño de las capas y el tamaño del kernel de convolución. Entrena diferentes redes de 2 y 3 capas con tamaños para la primera capa 2, 4 y 8. Para el resto ve doblando el tamaño de la primera capa para las capas sucesivas. Para los tamaños del kernel de convolución usa 3 y 5. Para crear la red solo tenienes que crear un objeto de la clase `convolutional` pasándole los parámetros adecuados. Al generar el objeto tienes que subir el modelo a la GPU usando el método `modelo.to('cuda')` (si no torch se quejará de que los datos y el modelo no están en el mismo dispositivo). Verás que la función de entrenamiento retorna la historia de la función de pérdida para el entrenamiento y validación. Usa esta información para comprobar la sobre especialización de la red y el número de épocas que realmente se realizan antes de parar. Puedes dejar la paciencia en su valor por defecto.

c) Verás en la implementación de la red que se usan una capa de pooling. Estas capas reducen el tamaño de la entrada haciendo una operación sobre grupos de valores vecinos. Por defecto usa `MaxPool2d` reduciendo el tamaño de la entrada a la cuarta parte (la mitad en dos dimensiones) usando el valor máximo de activación de cuatro valores contiguos. Otra posibilidad es usar la media de los valores con la capa `AvgPool2D`. Ajusta redes con el número de capas y el tamaño de kernel que te dio mejor resultado en el apartado anterior usando esta capa de pooling. Puedes evaluar la sobre especialización del modelo y evaluarlo de la misma manera que en el apartado anterior. Compara los resultados.

d) Habrás visto que las implementaciones de las redes tienen dos capas opcionales `BatchNorm2D` y `Dropout`. La primera utiliza estadísticas de las activaciones calculadas a partir de los datos para normalizarlas durante el entrenamiento, esto estabiliza el entrenamiento y actúa como una regularización. La segunda hace cero de manera aleatoria las activaciones que le llegan. Esto evita que la red dependa específicamente de alguna de las entradas que le llegan a las capas y actúa también como una regularización.

Repite el entrenamiento del apartado anterior escogiendo la capa de pooling que funcionó mejor activando `BatchNorm2D`. Haz lo mismo usando la capa de dropout usando como valores 0.05 y 0.01. Tendrás que aumentar la paciencia en el entrenamiento un poco para estos experimentos para que no acaben enseguida. Ahora podemos ver el efecto de combinar las dos capas, ajusta las redes usando `BatchNorm2D` y el mejor valor de dropout. Compara los resultados y comenta las diferencias que hayas visto en como evoluciona la función de pérdida usando estas regularizaciones y el efecto en el número de épocas que hace la red.

3. **Formas de ser normal**

   El preproceso de los datos tiene una gran influencia en el resultado cuando pasamos a trabajar con métodos no lineales. Esto es más patente en datos no tabulares. En este problema trabajaremos con un conjunto de datos compuesto por fragmentos de obras de 20 autores seleccionados[4]

   La tarea corresponde a averiguar si un texto proviene de la lengua original (inglés) o es una traducción. El conjunto de datos son fragmentos extraídos al azar de obras de autores de diferentes épocas y géneros con el objeto de comprobar si realmente se puede realizar esta clasificación. Todos están escritos en inglés, pero algunos de ellos han sido traducidos de otros idiomas, como el ruso, el francés o el italiano entre otros.

   Los datos los podéis obtener mediante la función `load_translation` de la librería `apafib`. Esta función retornará una lista con los textos y otra con las etiquetas que les corresponden. Resuelve los siguientes apartados ilustrando los resultados de la manera que te parezca más adecuada.

   a) El trabajar con texto es algo diferente de los datos tabulares habituales. En este tipo de datos se obtiene la matriz de datos a partir de extraer un subconjunto de las palabras de los textos y hacer un cálculo sobre ellas como por ejemplo determinar si están o no en un ejemplo o el número de veces que aparece. Esto lo podemos hacer con la función `CountVectorizer` del scikit-learn que es precisamente para este tipo de datos. Generaremos una matriz de datos con 1000 palabras[5], ya que nos imaginamos que decidir en este problema estará en los detalles. Podemos generar la matriz de datos para que contenga el número de veces que aparecen. Emplea el parámetro `stop_words` que tiene esta función indicando que el idioma del texto es el inglés.

   Fíjate que esto **es un preproceso** como otros que hemos ido empleando, así que aplícalo correctamente a los datos. Genera una partición de entrenamiento y una de test (80 %/20 %) que sea estratificada (fija también el estado del generador de números aleatorios para la reproducibilidad).

   Aplica reducción de dimensionalidad usando PCA y t-SNE. ¿Se puede ver alguna separabilidad entre los datos en la proyección en 2D? Comenta los resultados.

   b) Tenemos datos que corresponden a distribuciones multinomiales (son conteos). Si asumimos que las palabras de un texto son independientes (no lo son realmente) podemos usar Naïve Bayes para clasificarlos y usarlo como resultado base. Aplica este método a los conjuntos de datos que has generado y determina la calidad del modelo. Comenta los resultados.

   c) Tenemos bastantes opciones para normalizar los datos, podemos simplemente no hacerlo, podemos usar estandarización, podemos usar escalado o podemos usar normalización vectorial usando la norma L_1 y L_2. Explica que hace cada una de las normalizaciones.

[4] Los textos se han extraído de libros mantenidos por el proyecto Gutemberg.

[5] Fijate que este modelo tiene un parámetro `max_features` que te permite escoger el tamaño del vocabulario.

Entrena modelos de K-vecinos cercanos usando estas normalizaciones y explorando adecuadamente sus hiperparámetros. compara los resultados de los mejores modelos para cada normalización.

d) Ahora ajustaremos un clasificador MLP usando las mismas normalizaciones. Explorando adecuadamente sus hiperparámetros. Compara los resultados de los modelos ¿son consistentes los resultados de las normalizaciones? Es decir, ¿la misma normalización funciona mejor para los dos modelos?

4. **De esta nos hacemos ricos**

Las finanzas son un área de aplicación del aprendizaje automático y las redes neuronales. Esta no es sencilla dados los múltiples factores que intervienen y lo difícil que es representarlos y estimarlos adecuadamente, pero a veces se pueden hacer estimaciones que dan una idea del comportamiento de los datos.

En este problema trabajaremos con datos reales del NASDAQ sobre la cotización de las acciones de cinco empresas tecnológicas (Google, Microsoft, Apple, Intel y AMD) durante cinco años. Puedes obtener los datos mediante la función `load_Google` de la librería `apafib`. Esta retornará un dataframe que tiene tres columnas para cada acción, el valor final de la acción al final del día (P), el volumen de acciones que se intercambiaron (V) y la diferencia entre el mayor y el menor precio al que cotizaron en el día (GAP). En este problema intentaremos predecir la evolución del precio de la acción de Google (`GOOGLE-P`).

Resuelve los siguientes apartados ilustrando los resultados de la manera que te parezca más adecuada.

a) Haz un estudio de las características de los datos calculando sus estadísticas, la correlación entre los datos y representándolos de la manera que te parezca interesante. ¿Crees que puede ser posible predecir unas variables a partir de otras?

Divide los datos en conjunto de entrenamiento (los 1000 primeros días), otro de validación (100 días) y test (el resto). Estandariza los datos.

Para generar los datos para los modelos necesitaremos ventanas temporales de cierta longitud. `Pandas` permite generar una copia de una tabla de datos desplazada una serie de instantes temporales usando el método `shift`. Genera un conjuntos de datos con longitud de ventana 8 de manera que puedas predecir un día a partir de las variables de los siete anteriores Elimina todos los valores perdidos que ha generado esta transformación.

Para pasarle los datos a un modelo de `torch` hay que definir una clase del tipo `Dataset`, tienes una definida en el notebook del problema a la que hay que pasarle la matriz de datos y el vector con los valores a predecir. Esta clase se ha de pasar a un objeto `Dataloader` que se encarga de organizar el uso de los datos durante el entrenamiento, entre otras cosas de partir los datos en grupos (`batch_size`), puedes ver un ejemplo en el notebook.

b) Tenemos un problema de regresión, podemos usar un MLP que tenga una salida lineal para hacer la predicción con varias capas ocultas y ajustarlo usando mínimos cuadráticos (mean squared error). En el código del notebook tenéis una función que permite generar modelos MLP variando el número de capas y la función de activación. La red se define como una clase python donde en el método `__init__` definimos los elementos de la red y en el `forward` como se hace la propagación hacia adelante en la red (la propagación hacia atrás se calcula automáticamente).

Para entrenar los modelos usaremos la función `train_loop` que tienes en el notebook. Esta función recibe el modelo, el optimizador a usar, los datos de entrenamiento y validación,

la paciencia para la terminación temprana y el número de iteraciones que hará el entrenamiento. Para el optimizador usaremos `adam`, puedes ver en el notebook como generar el objeto del optimizador.

Entrena diferentes redes de 1 y 2 capas con diferentes tamaños usando como función de activación `Sigmoid` y `ReLU`. Para los tamaños de las dos capas puedes elegir el tamaño de la primera capa oculta y después reducirlo a la mitad para la segunda capas. Explora también la tasa de aprendizaje.

Para crear la red solo tienes que crear un objeto de la clase `MLP` pasándole los parámetros adecuados. Al generar el objeto tendras que subir el modelo a la GPU usando el método `modelo.to('cuda')` (si no torch se quejará de que los datos y el modelo no están en el mismo dispositivo). Verás que la función de entrenamiento retorna la historia de la función de pérdida para el entrenamiento y validación. Usa esta información para comprobar la sobre especialización de la red y el número de épocas que realmente se realizan antes de parar.

Evalúa los resultados usando MAE y MSE. Para ello puedes usar la función `test_model` que tienes en el notebook. Se le ha de pasar el modelo entrenado y un conjunto de datos y retornará las predicciones y las reales, a partir de ahí puedes usar las funciones de `scikit learn` para obtener las medidas de evaluación.

c) Existen muchas funciones de activación más allá de las clásicas. Experimenta con las funciones de activaciόm `LeakyReLU` y `SiLU` con la red con el número de capas que haya tenido mejores resultados explorando el tamaño de las capas. Compara la calidad de los modelos.

d) Una cosa que se nos ocurriría es que información sobre la fecha, como el día de la semana, la semana del año, o el mes podrían tener una influencia en las cotizaciones. Verás que el índice del `DataFrame` de los datos es la fecha. Extrae estos valores y añádeselos a los datos. Ajusta el modelo de red con el número de capas y función de activación que mejor haya funcionado explorando diferentes tamaños en las capas. Compara los resultados.

e) Un modelo base que podemos utilizar para saber si realmente estamos prediciendo algo es el llamado modelo de persistencia. Para este problema sería usar el valor de la acción de Google del día anterior como predicción. Calcula el MAE y el MSE para este modelo y compáralo con los resultados que has obtenido con el MLP.