Tarea 3

Fecha de entrega: 26 de junio 2023

Profesor: Felipe Tobar
Auxiliares: Catherine Benavides, Camila Bergasa, Víctor Caro,
Camilo Carvajal Reyes, Diego Cortez M., Stefano Schiappacasse

Formato de entrega: Reporte en formato PDF con extensión máxima de 7 páginas, presentando y analizando sus resultados, y detallando la metodología utilizada. Recomendación: Incorpore gráficos para apoyar su análisis y utilice formato doble columna si le falta espacio. Adicionalmente, debe entregar el código generado con la resolución de la tarea. Fije una semilla con su rut sin el dígito verificador para poder replicar sus resultados.

Parte 1: *Clustering* y reducción de dimensionalidad - 40 %

El siguiente *dataset* contiene 5000 imágenes de rostros de celebridades. Para cargarlo hacer click en este enlace, descomprimirlo y ejecutar el código siguiente.

```python
import numpy as np
import os
from PIL import Image

# Cargamos las imágenes del dataset
imgs = np.array([np.array(Image.open("images/"+file)) for file in os.listdir("images")])

# Cargamos las imágenes de test
test_images = np.array([np.array(Image.open(f"test/test_{i}.jpg")) for i in range(1, 6)])

# Pasamos cada imágen a formato vector
data = imgs.reshape(5000, 44*28)
```

Usted deberá explorar el *dataset* y aplicar técnicas de reducción de dimensionalidad y *clustering* para analizarlo. Puede utilizar librerías como *sklearn* para esto. Debe elegir si normalizar o no antes de aplicar reducción de dimensionalidad, justificando **en cualquiera de los casos**.

(a) (1,0 pts) Aplique PCA para reducir la dimensión de los datos y visualice los primeros 10 componentes y su media, reordenando como imagen con `.reshape(44, 28)`. Comente lo que observe en función de lo que representa cada componente principal.

(b) (2,0 pts) Calcule la raiz del error cuadrático medio (RMSD) entre las caras y la reconstrucción de ellas en todo el conjunto de datos, utilizando una cantidad variable (al menos 20 distintas) de componentes principales entre 2 y el máximo posible. Análogamente, reconstruya las imágenes

del conjunto de test y calcule el RMSD de cada una de ellas y sus reconstrucciones en función del número de componentes principales. Grafique tanto el RMSD de la reconstrucción de los datos del conjunto principal como cada uno de los RMSD de los datos de test (por separado) en función de la cantidad de componentes principales (6 líneas). Comente lo que observa y responda las siguientes preguntas:

- ¿Qué puede decir sobre lo que sucede con la cantidad máxima de componentes principales?
- ¿Qué puede decir sobre lo que sucede con cantidades intermedias y bajas de componentes principales?
- ¿Que conclusiones puede sacar sobre cómo PCA captura la estructura del espacio de los datos?

(c) (1,5 pts) Elija sólo los pixeles correspondientes a la boca (`[30:40, 5:23]`) y realice *clustering* con ellos mediante KMeans, utilizando 20 *clusters*. Grafique la media de las caras que pertenecen a cada *cluster* reordenando como imagen. Elija el *cluster* que más le llame la atención y grafique la proyección en 2 dimensiones (de la cara completa) mediante un *scatterplot*, coloreando por su *cluster* elegido. Luego remplace uno de los componentes principales por la media del *cluster* menos la media de los datos, proyecte y grafique. Comente lo que observa y responda ¿Cómo lo haría para aumentar el grado de separación del *cluster* en la dimensión proyectada?

Para la siguiente actividad, utilice el *dataset Fashion MNIST*. Para descargarlo, ver el código presente en el anexo 1.

(d) (1,5 pts) Realice *clustering* mediante KMeans en los datos originales utilizando 10 *clusters*. Luego, reduzca la dimensión de los datos mediante PCA a 2 dimensiones y repita el procedimiento anterior. Grafique en el espacio proyectado coloreando por los *clusters* encontrados con el método 1, luego con el método 2 y luego por las clases reales de los datos. Comente lo que observa. ¿Cómo podría aumentar la separación de las clases en el gráfico de PCA?

Parte 2: Redes neuronales - 60 %

Consideramos un *dataset* $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ con $y^{(i)} \in \{c_1, \dots, c_K\}$ (i.e., consideramos un problema de clasificación multiclase). Dadas p y q dos distribuciones de probabilidad discreta, la **entropía cruzada** está dada por

$$H(p, q) = \sum_{j=1}^{K} p(c_j) \log\left(\frac{1}{q(c_j)}\right) .$$

Se define entonces la función de pérdida de entropía cruzada por

$$L(q, p) = \frac{1}{N} \sum_{i=1}^{N} H(p_i, q_i)$$

(a) (1,5 pts.) Demuestre que

$$\hat{\theta} = \arg\min_{\theta} L(p_\theta, p)$$

donde $\hat{\theta}$ es el estimador de máxima verosimilitud y p es la distribución de probabilidad empírica.

Consideraremos, para el caso particular de una clasificación binaria, un modelo en el cual modelamos la probabilidad de que un elemento pertenezca a la clase 1 está dada por

$$p_\theta(y=1|x) = \sigma(\sum_{j=1}^{d} x_j w_j + b)\,,$$

donde $\sigma(u) = \frac{1}{1+e^{-u}}$ es la función de activación y $\theta = (w_1, \ldots, w_N, b)$ son los parámetros del modelo (d corresponde a la dimensionalidad del input x).

(b) (1.5 ptos.) Plantee una manera de entrenar este modelo con alguna pérdida adecuada que involucre el output real $y^{(i)}$ de un par $(x^{(i)}, y^{(i)})$ y que use el algoritmo de gradiente estocástico (SGD). Calcule las derivadas parciales de para algún $w \in \{w_j\}_{j=1}^{d}$ arbitrario y b. Use esto para expresar las actualizaciones de (SGD) de manera explícita. Mencione que cambiaría si agregamos una representación intermedia (agregar una capa) en el algoritmo anterior. Explique de manera simple la importancia y funcionamiento del algoritmo *backpropagation* dado todo lo anterior.

En lo que sigue usaremos la base de datos *Fashion MNIST*, que consiste en imágenes en blanco y negro de artículos de tiendas de ropa. La tarea consiste en entrenar un modelo que identifique de manera automática la clase de la imagen en cuestión (esto es, una clasificación multiclase). Las instrucciones para la descarga del *dataset* se encuentran el anexo 1. El primer objetivo será definir un modelo que conste de una sola capa lineal, lo cual equivale a una regresión logística si le aplicamos la función *softmax*. Definimos los parámetros de nuestra regresión como tensores, los cuales serán después optimizados.

(c) (1 pto.) Implemente **Descenso de Gradiente Mini-batch** para ajustar los parámetros del modelo dada una tasa de aprendizaje. Implemente una función entrenar que tome un número de épocas y ejecute el método anterior en cada iteración, imprimiendo el valor de la función de pérdida. Guarde además estas pérdidas en una lista. (A una barrida completa al conjunto de entrenamiento se le llama época (o *epoch* en inglés).) Ejecute el bucle para 10 épocas y grafique lo encontrado. Utilice otras métricas para evaluar la calidad de la clasificación, ¿que clases son más y menos fáciles de reconocer con el algoritmo?

```
def DescensoGradiente(pesos,sesgo,tasa):
    perdidas_epoch = []
    # rellenar acá
    cantidad_batches = None  # cambiar
    # ------------
    for i in range(cantidad_batches):
        # rellenar acá
        loss = None  # calcular pérdida
        # ------------
        loss.backward()
        with torch.no_grad():
            pesos -= pesos.grad * tasa
            sesgo -= sesgo.grad * tasa
            pesos.grad.zero_()
            sesgo.grad.zero_()
        perdidas_epoch.append(loss.item())
    return perdidas_epoch

```

```
logsoftmax = torch.nn.LogSoftmax(dim=0)

def predicciones(output):
    return torch.argmax(logsoftmax(output),axis=1)
```

(d) (1 pto.) Re-defina el método entrenar para que tome un optimizador del módulo *optim*, un modelo (sin entrenar) y un número de épocas. Programe y ejecute el método con Descenso de Gradiente (SGD) del del modo usual en *pytorch* y grafique los resultados. Escoja otro optimizador del módulo *optim*, explique brevemente las diferencias de tal optimizador con SGD entrene modelos con ellos y compare.

(e) (1 pto.) Un tipo de red neuronal que funciona bien para el procesamiento de imágenes son las **redes convolucionales (*CNN*)**. Investigue y mencione alguna ventaja de usar redes convolucionales. A continuación, implemente un modelo de red neuronal usando redes convolucionales y ReLu como no linealidad. Se recomienda colocar dos capas convolucionales que aumenten el número de canales y ejecutar un max-pooling entre ellas y la no-linealidad, finalizando por una capa lineal. Pruebe su red del mismo modo que antes y compare los resultados con las partes anteriores.

1. Anexo: Cargar el *dataset Fashion MNIST*

Para cargar el *dataset* dirigirse al siguiente sitio y descargar el archivo en cuestión (hacer click en *Download (72 MB)*). Una vez descargando y extrayendo los archivos estos quedarán en una carpeta. A modo de ejemplo colocamos esa carpeta en el mismo directorio del script/notebook y la llamamos *data/*. Para cargar los datos de entrenamiento desde el archivo como arreglos de *numpy* executamos.

```
import pandas as pd

train = pd.read_csv("data/fashion-mnist_train.csv")
y_train = train['label'].values
x_train = train[list(train.columns)[1:]].values
x_train = x_train.reshape(-1, 784)
x_train = x_train / 255
```

En el código anterior hemos dejado los datos dentro del intervalo $[0, 1]$ (de la escala de gris) de cada pixel y hemos covertido la matriz de input 28×28 que representa cada imagen en un vector plano. Finalmente para convertir el *dataset* a tensores de *pytorch* debemos ejecutar:

```
import torch

x_train, y_train = map(torch.tensor, (x_train, y_train))
x_train = x_train.float()
```