Universidad de
SanAndrés

I302 - Aprendizaje Automático y Aprendizaje Profundo

1er Semestre 2025

Trabajo Práctico 3

Fecha de entrega: Lunes 12 de mayo, 23:59 hs.

Formato de entrega: Los archivos desarrollados deberán entregarse en un archivo comprimido (.zip) a través del Campus Virtual, utilizando el siguiente formato de nombre: *Apellido_ Nombre_ TP3.zip*. Se aceptará únicamente un archivo por estudiante y debe contener por lo menos los siguientes elementos:

```
Apellido_Nombre_TP3.zip/
|- data/
|- Apellido_Nombre_Informe_TP3.pdf
|- Apellido_Nombre_Notebook_TP3.ipynb
|- predicciones.csv
```

- **Informe:** Debe incluir todos los aspectos teóricos, decisiones metodológicas, visualizaciones, análisis y conclusiones. El objetivo es que el informe contenga toda la explicación principal del trabajo. Se puede hacer referencia al notebook con frases como "Ver sección X del notebook para la implementación". El informe debe entregarse utilizando el archivo `template_informe.tex` provisto y no debe exceder las 10 páginas.
- **Notebook:** Debe contener el código utilizado, experimentos, análisis exploratorio, gráficos y el proceso completo de procesamiento y modelado. Sirve como respaldo técnico del informe y debe estar ordenado y bien documentado. Se recomienda modularizar el código en archivos `.py` cuando sea posible.

Trabajo Práctico: Redes Neuronales

El objetivo de este trabajo es desarrollar y evaluar modelos basados en redes neuronales, incorporando técnicas de ablación para entender el impacto de diversas modificaciones en el proceso de entrenamiento y en la capacidad de generalización del modelo. No se permite usar librerías de machine learning como scikit-learn o PyTorch, a menos que sea pedido explícitamente en el enunciado del ejercicio.

1. **Análisis y Preprocesamiento de Datos**

   - El dataset que vamos a utilizar es similar a MNIST pero con caracteres japoneses. Son imágenes de 28×28 con 49 clases posibles.
   - Los datos se pueden abrir con:

```
import numpy as np

X_images = np.load("X_images.npy")
y_images = np.load("y_images.npy")
```

   - Examinar el dataset y visualizar al menos 3 imágenes. Para crear la matriz y graficar la imagen, usar el comando:

```
img = X[0].reshape(28,28)
```

   - Dividir el conjunto de datos en tres subconjuntos: **Train**, **Validation** y **Test**.
   - Dividir todos los valores por 255, para que el máximo sea 1.

2. **Implementación y Entrenamiento de una Red Neuronal Básica**

   - Implementar una red neuronal con L capas ocultas cada con $M^{(l)}$ nodos con función de activación **ReLU** (en las capas ocultas) y activación **softmax** para la capa de salida.
   - Implementar un algoritmo para entrenar dicha red, mediante backpropagation y gradiente descendiente estándar, utilizando como función de costo la cross-entropy.

     NOTA: El algoritmo backpropagation debe sera adaptado para el caso de clasificación multi-clase con función de activación **softmax** para la capa de salida y función de costo cross-entropy.
   - Entrenar una red neuronal con 2 capas ocultas, con 100 y 80 nodos respectivamente, y graficar la evolución de la función de costo (cross-entropy) sobre los conjuntos de entrenamiento y validación a lo largo de las épocas. Llamaremos a este modelo **M0**.
   - Reportar las siguientes métricas de performance, sobre los conjuntos de entrenamiento y validación, para el modelo base entrenado:
     - Accuracy
     - Cross-Entropy
     - Matriz de Confusión

3. **Implementación y Entrenamiento de una Red Neuronal Avanzada**

   - Implementar las siguientes mejoras al algoritmo de entrenamiento, y para cada una reportar el efecto observado sobre el tiempo de entrenamiento y la performance del modelo resultante.
     - Rate scheduling lineal (con saturación) y exponencial.
     - Mini-batch stochastic gradient descent.
     - Optimizador ADAM.
     - Regularización L2.
     - Regularización mediante early stopping.
     - OPCIONAL: Regularización mediante dropout.
     - OPCIONAL: batch normalization.
   - Explorar cambios en la arquitectura de la red (es decir, la cantidad de capas ocultas y unidades ocultas por capa), y los hiperparámetros (cada uno de los items en la lista anterior tiene una serie de parámetros que podemos variar), y determinar la configuración que funcione mejor (menor error de validación). Llamaremos a este modelo **M1**.

4. **Desarrollo de una Red Neuronal con PyTorch**

   - Utilizando PyTorch, entrenar una red neuronal con la arquitectura y los hiperparámetros hallados en el ejercicio anterior. Llmaremos a este modelo **M2**.
   - Utilizando PyTorch, explorar cambios en la cantidad de capas ocultas y unidades ocultas por capa, y determinar la configuración que funcione mejor. LLamaremos a este modelo **M3**.
   - Utilizando PyTorch, encontrar una arquitectura (capas ocultas y unidades por capa) que produzca overfitting. Llamaremos a este modelo **M4**.
   - Comparar la performance sobre el conjunto de test de los siguientes cuatro modelos:
     a) El modelo base de implementación propia (**M0**).
     b) La mejor arquitectura obtenida con la implementación propia (**M1**).
     c) Modelo en PyTorch, usando la misma arquiterctura e hiperparámetros que en la implementación propia (**M2**).
     d) La mejor arquitectura obtenida en PyTorch (**M3**).
     e) Una arquitectura en PyTorch con sobreajuste (**M4**).

5. **Desafío**

   - Utilizando el modelo que considere que sea el mejor, predecir las probabilidades a-posteriori de cada clase del dataset *X_COMP.npy* y generar un archivo .csv con las predicciones llamado *Apellido_Nombre_predicciones.csv*.

- El archivo *predicciones.csv* debe tener una fila por muestra y las columnas deben ser las probabilidades a posteriori de cada clase.

  Ejemplo del formato esperado:

Clase_0	Clase_1	Clase_2	$\cdots$	Clase_47	Clase_48
0.0012	0.0000	0.0005	$\cdots$	0.0021	0.0000
0.0000	0.0003	0.0010	$\cdots$	0.0000	0.9982
0.0006	0.0000	0.0000	$\cdots$	0.9810	0.0001
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$