Universidad de SanAndrés

I302 - Aprendizaje Automático y Aprendizaje Profundo

2do Semestre 2025

Trabajo Práctico 3

Fecha de entrega: Miercoles 29 Octubre, 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 **EMNIST Bymerge**, un conjunto extendido de MNIST que incluye dígitos y letras manuscritas. Son imágenes de 28×28 píxeles con **47 clases posibles**. Algunas letras mayúsculas y minúsculas se encuentran agrupadas cuando son indistinguibles (por ejemplo, C/c, O/o).
Los datos se pueden abrir con:

```
import numpy as np
X_images = np.load("X_images.npy")
y_images = np.load("y_images.npy")
```

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

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

b) Dividir el conjunto de datos en tres subconjuntos: Train, Validation y Test.

c) Normalizar dividiendo todos los valores por 255, de modo que el máximo sea 1.

2. Implementación y Entrenamiento de una Red Neuronal Básica

a) Implementar una red neuronal Multi-layer perceptron (MLP) con L capas ocultas, cada una con $M^{(l)}$ nodos, utilizando ReLU en las capas ocultas y activación softmax en la capa de salida.

b) Implementar un algoritmo para entrenar dicha red, mediante backpropagation y gradiente descendente estándar, utilizando como función de costo la cross-entropy.

NOTA: El algoritmo backpropagation debe adaptarse al caso de clasificación multiclase con función de activación softmax en la salida y función de costo cross-entropy.

c) Entrenar una red neuronal con 2 capas ocultas, con 128 y 64 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 M_0.

d) Reportar las siguientes métricas de performance, sobre los conjuntos de entrenamiento y validación, para el modelo base entrenado:

- Accuracy
- Cross-Entropy
- Matriz de Confusión
- F1-Score Macro

3. Implementación y Entrenamiento de una Red Neuronal Avanzada

a) 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 (L_2, Early Stopping)
- Opcional: Batch normalization.
- Opcional: Label smoothing.
- Opcional: Gradient clipping.

b) Explorar cambios en la arquitectura de la red (es decir, la cantidad de capas ocultas y unidades ocultas por capa), y en los hiperparámetros (cada uno de los ítems en la lista anterior tiene parámetros que se pueden variar), y determinar la configuración que funcione mejor (menor error de validación). Llamaremos a este modelo M_1.

4. Desarrollo de una Red Neuronal con PyTorch

a) Utilizando PyTorch, entrenar una red neuronal con la arquitectura y los hiperparámetros hallados en el ejercicio anterior. Llamaremos a este modelo M_2. Comparar la performance del modelo M_1 y M_2 para validar que los comportamientos de los modelos son parecidos.

b) Utilizando PyTorch, explorar cambios en la cantidad de capas ocultas y unidades ocultas por capa, y determinar la configuración que funcione mejor. Además de la arquitectura, se deberá experimentar con funciones de activación más modernas disponibles en `torch.nn`, tales como `LeakyReLU`, `SiLU`, `Swish` o `GELU` y estrategias de regularizacion adicionales como Dropout (ademas de las ya mencionadas previamente). Llamaremos a este modelo M_3.

c) Comparar la performance sobre el conjunto de test de los siguientes cuatro modelos:

   1) El modelo base de implementación propia (M_0).
   2) La mejor arquitectura obtenida con la implementación propia (M_1).
   3) Modelo en PyTorch, usando la misma arquitectura e hiperparámetros que en la implementación propia (M_2).
   4) La mejor arquitectura obtenida en PyTorch (M_3).

d) Con los modelos ya entrenados, perturbar los datos de test con distintos niveles de ruido y evaluar cuan robusta es la performance de cada modelo frente a estos datos perturbados en comparacion a los resultados obtenidos de datos sin perturbar.