Apuntes Semana 9 – 24/04/2025

Marco Rivera Serrano

***Abstract*—En esta clase, se habla sobre artículos acerca de nuevos modelos de lenguaje de ChatGPT y despidos de Intel. Además, un sondeo sobre cómo va el proyecto I y la explicación del proyecto II así como sus especificaciones. Por otro lado, se hace un repaso de la clase pasada sobre backpropagation, operaciones con grafos, derivadas, entre otros temas. Por último, se hace una explicación de dos notebooks. El primero utiliza PyTorch para diseñar una arquitectura de red nuronal. El segundo, es un notebook donde se implementa una red neuronal desde cero y se deja una tarea para corregir el gráfico.**

I. Noticias sobre Inteligencia Artificial

A. Paper sobre el nuevo modelo ed ChatGPT

Nombre: OpenAI o3 and o4-mini System Card. (Link)

Este artículo habla sobre el modelo *OpenAI o3* y *o4-mini* que combinan el razonamiento con capacidades de herramientas. Estos modelos sobresalen en la solución de problemas matemáticos complejos, codificación, percepción y análisis visual.

B. Despidos de Intel

Según lo hablado en clase, Intel hizo un recorte masivo de más del 20% de su plantilla.

II. Sondeo del proyecto I

El grupo presenta cierta incertidumbre a la hora de contestar, pero están avanzados en el proyecto. Sin embargo, presentan dudas sobre la fecha de entrega.

III. Explicación del proyecto II

Este proyecto tiene como fecha de entrega el 15 de mayo. Su objetivo es realizar el procesamiento de señales a partir de imágenes (espectograma). Se menciona que una red completamente conectada (fully-connected) es muy cara para las imágenes. Ejemplo de ello es en NMIST de letras.

Para poder cumplir con el objetivo, se tiene que aplicar las redes neuronales convolucionales (CNN) para la clasificación multiclase utilizando aprendizaje supervisado. Además, se debe hacer uso de MachineLearning y PyTorch asimismo como herramientas de monitorero. La idea es utilizar los audios como espectogramas para la red convolucional. Para obtener las etiquetas, se recomienda utilizar el directorio de las imágenes. La red neuronal debe detectar los patrones en el espectro y ver cómo predice el número que representa dicho espectro. Si el grupo de trabajo no cuenta con una GPU, se recomienda el uso de Google Colab. Además, se debe subir el dataset a Google Drive y conectarse entre sí. Dentro de la especificación del proyecto, hay una guía para este proceso.

A. Arquitectura del proyecto

Se deben crear dos modelos de forma manual (no utilizar modelos predefinidos) utilizando PyTorch. El modelo A, utiliza la arquitectura LeNet-5 el cual altera el tamaño y los hiperparámetros. El modelo B, queda a decisión de los estudiantes los cuales pueden implementar alguna arquitectura de la literatura o basarse en ella para crear una propia.

B. Procesamiento de imágenes

En esta parte del proyecto, se deben entrenar ambos modelos con dos conjuntos de datos distintos. El primer conjunto deben ser los espectogramas crudos (sin filtros). El segundo debe aplicarse el filtro *Bilateral Filter*. Se recomienda no hacerlo de forma paralela con el entrenamiento.

C. Data Augmentation

Se debe aplicar en el dataset si presenta clases sesgadas y debe justificar la elección del mismo.

D. Entrenamiento

Ambos modelos deben utilizar cuatro datasets:

- Crudo y sin data augmentation.
- Filtro bilateral sin data augmentation.
- Crudo con data augmentation.
- Filtro bilateral con data augmentation.

Dando un total de ocho entrenamientos.

De la misma manera, el dataset se debe dividir en tres partes (entrenamiento, validación y prueba) y verificar si los modelos no presentan sobre o subajuste.

E. Evaluación de modelos

Al comparar el comportamiento de ambos modelos, se recomienda utilizar la herramienta *Weights and Biases*, la cual permite la visualización de diferentes aspectos durante el entrenamiento.

F. Entregables

Se deben subir los siguientes archivos:

- Jupyter(s) completo.
- Informe escrito.

G. Notas

Hay un concurso de mejora continua. Cada equipo tiene tres oportunidades según el F1 - Score y para participar, solo se deben enviar los resultados finales de todas las métricas. El equipo ganador conseguirá 10 puntos extra.

Por otro lado, al usar Google Colab, se debe tener cuidado con los datos tipo CUDA.

IV. Repaso de la clase pasada

Se inica la clase con el concepto de *back propagation*, el cual corresponde a ajustar los parámetros desde la última capa a la primera. Las redes son como grafos y pueden realizarse operaciones matemáticas. Esto permite que exista alta complejidad o simplicidad. Así, se pueden paralelizar cálculos pesados.

Con respecto a las neuronas, el super índice indica la capa, mientras que el subíndice indica la neurona. Por otro lado, los pesos tienen dos subíndices: j, es la neurona que pertenece y k, de dónde proviene. Con esto, se construye una matriz de los pesos.

Cabe resaltar que las neuronas tienen el *bias* y la neurona se encarga de realizar la suma de las salidas de la capa anterior para poder aplicar el producto punto. Esto implica tener mucho cuidado con los índices para manejar los datos correctos. Cabe resaltar que cada neurona tiene un *bias* diferente, entonces cada operación en sí, también lo será.

Por otra parte, cada neurona tiene que actualizar sus parámetros y tiene un comportamiento local. Cabe resaltar que las derivadas son obtenidas de la función de pérdida y tienen más índices con respecto a la capa anterior.

V. Explicación de los Notebooks

A. Introducción a PyTorch

Este es el archivo llamado `Redes_neuronales.ipynb`. Su objetivo es familiarizar a los estudiantes con el uso de PyTorch y las redes neuronales en código. Este modelo utiliza el set de datos MNIST. Ademas, se usa un modelo totalmente conectado y tiene las siguientes capas:

- **Flatten**: Aplana la imagen 2D a un vector 1D.
- **Linear**: Primera capa con una función de activación *ReLU*.
- **Linear**: Segunda capa, igualmente con ReLU.
- **Linear**: Salida con las 10 clases.

Se definen los hiperparámetros esenciales como el tamaño del lote, learning rate (α) y cantidad de épocas. Luego, se cargan los datos del dataset y se transforman en tensores. Estos datos se normalizan y transforman para un mejor manejo.

1) Entrenamiento: Se utiliza un bucle para las épocas, en donde se utiliza la función `model.eval()` la cual permite evaluar el modelo. Por otra parte, en la evaluación no se calculan los gradientes, dado que no es necesario actualizar los pesos en esta fase. Sin embargo, en el entrenamiento sí se activan dado que se necesitan ajustar e ir propagando el error con la función `loss.backward()`.

B. Red neuronal desde cero

Este archivo llamado `NN-From-Scratch.ipynb`, fue realizado por el profesor José Carranza.

1) Definición del framework: Primero, se definen funciones matemáticas que se utilizan en redes neuronales. Estas funciones son:

- `sigmoid(x)`: Devuelve el valor entre 0 y 1.
- `sigmoid_grad(sigmoid)`: Derivada de la función sigmoide.
- `relu(x)`: Función de activación ReLU.
- `relu_grad(x)`: Derivada de la función ReLU.
- `softmax(x)`: La función softmax en sí para la salida.
- `logloss(x, y)`: Función de pérdida de entropía cruzada.

2) Clase `base`: Es una plantilla para las capas de la red y permite funcionalidades que tendrán todas las capas. El constructor toma la cantidad de features de la entrada y la cantidad de clases de la salida. La función `forward(self, x)` permite la forma en que la capa computa su salida. La función `backward(self, prev)` permite la retro propagación y calcula los gradientes de los parámetros de la capa según la pérdida. Por último, la función `update(self, lr)` actualza los pesos y el sesgo según los gradientes calculados.

C. Clase `loss_layer`

Esta clase se centra en el cálculo de la función de pérdida y la retro propagación para la capa de salida de la red. La función `forward(self, x)` guarda la entrada, calcula una transformación lineal y aplica el softmax a la salida para obtener las probabilidades de salida. La función `backward(self, y)` convierte las etiquetas a un vector *one-hot* y calcula el gradiente para la retro propagación. Finalmente, la función `loss(self, y)` calcula la entropía cruzada que mide la diferencia entre las probabilidades predichas y las etiquetas.

D. Clase `Model`

Esta clase encapsula toda la red. La función `forward(self, x, y)` realiza el paso hacia adelante en toda la red. La función `backward(self, y, o)` realiza la retro propagación para calcular los gradientes de toda la red. Por último, la función `update(self, lr)` actualiza todos los pesos y sesgos en todas las capas.