Apuntes de Clase Semana 10 - 24/04/25

Israel Herrera Campos
IC6200 Inteligencia Artificial
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
israelhercam@estudiantec.cr

I. Información compartida de forma asincrónica el martes 22/04/25

1) Temas que se verán conforme se avance en el curso:

- Procesamiento de señales a partir de imágenes.
- Redes neuronales convolucionales (CNN), cómo se comportan y cómo se aplican las convoluciones. En términos básicos es una especie de filtro. Se tiene un filtro de 3x3, se aprenden los valores que van en el filtro (los cuales se convierten en nuestros pesos) y se desliza el filtro por la imagen para extraer características de la imagen.
- Continuación de clasificación multiclase.

2) Enunciado del Proyecto 2:

a) Descripción General: El objetivo del proyecto es aplicar redes neuronales convolucionales en la clasificación multiclase para el procesamiento de imágenes utilizando un aprendizaje supervisado. Se desarrollará un clasificador de dígitos empleando el dataset MNIST Audio [1], que contiene 30000 ejemplos de dígitos hablados del 0 al 9. Los datos fueron recopilados de 60 personas con diferentes acentos, géneros y edades. Aunque el dataset está en formato de audio, los modelos a implementar trabajarán con imágenes. Para hacer esto posible, se crearán espectrogramas de los audios, que son representaciones visuales de los patrones sonoros y servirán como entradas para los modelos CNN.

b) Arquitecturas de Modelos: **Modelo A:** Se implementará siguiendo una solución clásica basada en la arquitectura LeNet-5 [2], permitiendo modificaciones en el tamaño de las entradas y los hiperparámetros para optimizar su rendimiento. **Modelo B:** Para este modelo se puede implementar cualquier arquitectura documentada en la literatura o desarrollar una propia basada en investigaciones previas. Las arquitecturas recomendadas incluyen EfficientNet [3], MobileNet [4], Inception [5], DenseNet [6] y ResNet [7].

c) Preprocesamiento de Datos: Ambos modelos serán entrenados con dos variantes de conjuntos de datos:

- **Espectrogramas crudos**: Sin ninguna modificación u optimización adicional.
- **Espectrogramas procesados**: Aplicando el Bilateral Filter [8]. Este filtro tiene como objetivo principal reducir el ruido mientras preserva los bordes de la imagen. A diferencia de los filtros de suavizado tradicionales, considera tanto la cercanía espacial de los píxeles como la similitud de sus valores, permitiendo suavizar regiones homogéneas mientras respeta las discontinuidades importantes en la imagen.

d) Aumentación de Datos: Se implementará una técnica de aumentación de datos inspirada en procesamiento de audio para incrementar el número de ejemplos disponibles para el entrenamiento, siguiendo metodologías similares a las presentadas en SpecAugment [9].

e) Proceso de Entrenamiento: Se realizarán un total de ocho entrenamientos, cuatro para cada modelo. Los conjuntos de datos a utilizar serán:

- Espectrogramas crudos sin aumentación de datos
- Espectrogramas con filtro bilateral sin aumentación de datos
- Espectrogramas crudos con aumentación de datos
- Espectrogramas con filtro bilateral con aumentación de datos

f) Evaluación de Modelos: Los resultados de ambos modelos serán comparados utilizando la plataforma "Weights and Biases" [10], que permitirá visualizar y analizar las métricas de rendimiento de manera efectiva.

g) Concurso de Rendimiento: Se llevará a cabo un concurso interno donde cada equipo tendrá tres oportunidades para mejorar su F1-score antes de la fecha de entrega final. El equipo que obtenga el mayor puntaje después de las tres oportunidades será el ganador del concurso, recibiendo 10 puntos extra en la calificación del proyecto.

II. Materia de la clase pasada - Back propagation

Este se encarga de propagar el error o el grado de error que se tiene en nuestra función de pérdida sobre nuestra red para ir optimizando nuestros parámetros. Este tiene su contraparte, llamada Forward propagation, en la cual la entrada es la que se propaga a través de la red para obtener una salida o etiqueta. Mientras que el back propagation toma la etiqueta o el error de la pérdida para propagarla hacia toda la red con todos los hiperparámetros.

A. Funciones como operaciones de grafos

Se brinda el ejemplo básico de cómo resolver una función compuesta de una multiplicación y suma. A partir de esto se pueden renombrar las áreas de las funciones para que sea más legible y entendible. Esto es importante para la realización de derivadas parciales.

$$f(x, y, z) = (x + y)z \tag{1}$$

$$q = x + y \tag{2}$$

$$f = qz \tag{3}$$

B. Opimización del grafo

Se ve el ejemplo de una red que posee tres capas, donde cada capa solo tiene una neurona. Se pueden observar las siguientes funciones en relación a la misma:

$$L_i = (a^l - y_i)^2 \tag{4}$$

$$z^l = w^l a^{l-1} + b^l \tag{5}$$

$$a^l = g(z^l) \tag{6}$$

El superíndice indica el número de capa, el L_i es la función de pérdida, la capa de activación a^l está conformada por una función no lineal y la z^l representa la linealidad que existe. Esta z^l tiene los pesos y la entrada de que proviene de la salida de la neurona a^{l-1}. Es muy importante destacar que cada neurona tiene sus propios pesos y bias. Para despejar o encontrar resultados de variables en estas funciones, se debe de utilizar la regla de la cadena que nos permite manejar las dependencias entre múltiples variables.

C. Vector gradiente

Las derivadas parciales vistas durante el curso también se conocen como gradientes. Hacer el cálculo de estas gradientes con todos los parámetros que hay en la red se conoce como cálculo de gradientes o vector gradiente.

$$\nabla L = \left[\frac{\partial L}{\partial w^{(1)}} \ \frac{\partial L}{\partial b^{(1)}} \ \vdots \ \frac{\partial L}{\partial w^{(n)}} \ \frac{\partial L}{\partial b^{(n)}} \right]$$

Esta notación representa el gradiente de una función de pérdida L con respecto a los parámetros de un modelo, específicamente los pesos (w) y sesgos (b) de cada capa. El gradiente contiene las derivadas parciales que indican la dirección de mayor incremento de la función L.

D. Múltiples neuronas

En los casos donde se manejan dos o más neuronas, el super índice se encarga de definir la capa de la neurona, mientras que el subíndice indica la neurona en específico. Los pesos tienen dos subíndices, el j que indica donde pertenece y el k que indica donde proviene. Con esto se puede obtener la matriz de pesos.

III. PyTorch y Redes Neuronales

A. Introducción a PyTorch

PyTorch es una biblioteca de código abierto utilizada en la implementación de modelos neuronales complejos.

B. Notebook "Redes_neuronales.ipynb"

Implementa una red neuronal con PyTorch para la clasificación de dígitos del dataset MNIST. Contiene importación de librerías, definición de una red completamente conectada con tres capas, configuración de hiperparámetros (batch_size=64, learning_rate=0.001, num_epochs=10), carga y preprocesamiento de datos, entrenamiento con evaluación periódica, visualización de resultados mediante gráficos de pérdida, y almacenamiento del modelo entrenado. Este notebook se encuentra en la plataforma del TecDigital, en los documentos del curso.

C. Estructura de la Red Neuronal

La red neuronal implementada tiene la siguiente arquitectura:

$$\text{Entrada} \to \text{FC1}(28 \times 28 \to 128) \to \text{ReLU} \tag{7}$$

$$\to \text{FC2}(128 \to 64) \to \text{ReLU} \tag{8}$$

$$\to \text{FC3}(64 \to 10) \to \text{Salida (Logits)} \tag{9}$$

donde FC representa una capa completamente conectada (Fully Connected) y ReLU es la función de activación utilizada.

D. Proceso de Entrenamiento

Algorithm 1 Proceso de entrenamiento de la red neuronal

```
for epoch ∈ {1, 2, ..., num_epochs} do
    Evaluar el modelo en datos de prueba
    Cambiar a modo de entrenamiento
    for cada batch de datos de entrenamiento do
        Realizar forward propagation
        Calcular pérdida
        Poner gradientes a cero
        Calcular gradientes (backward propagation)
        Actualizar parámetros (optimizer.step())
    end for
    Registrar y mostrar pérdidas y precisión
end for
```

E. Resultados

El modelo alcanza una precisión aproximada del 81% en el conjunto de prueba. La gráfica muestra que la pérdida de entrenamiento disminuye constantemente, mientras que la pérdida de prueba se estabiliza después de aproximadamente 6 épocas, lo que podría indicar un ligero sobreajuste.

F. Notebook "NN-From-Scratch.ipynb"

Implementa una red neuronal desde cero sin utilizar frameworks de alto nivel como PyTorch o TensorFlow, solo usando NumPy para la clasificación de dígitos del dataset MNIST. El cuaderno contiene la definición de funciones de activación (sigmoid, ReLU), implementación de capas base, densas y de pérdida, construcción de un modelo completo, carga y preprocesamiento del dataset MNIST, y entrenamiento con

evaluación periódica mostrando métricas de pérdida y precisión a través de gráficos. Al igual que su contraparte, puede ser encontrado en la plataforma del TecDigital.

G. Componentes Principales

El código implementa desde cero:

Funciones de activación (sigmoid, ReLU) y sus derivadas. Clase base para todas las capas. Capa densa (fully connected). Capa de pérdida (softmax + cross entropy). Modelo completo que gestiona las capas. Manejo de datasets (entrenamiento, validación, prueba). Sistema de registro y visualización de métricas. Inicialización de pesos con el método Xavier. Sistema de batch para entrenamiento eficiente.

H. Hiperparámetros

Los principales hiperparámetros configurados son:

batch_size = 32. learning_rate = 0.0035. epochs = 20. hidden_shapes = [512, 32]. validate_every_no_of_batches = 600.

La implementación destaca por construir todas las partes de una red neuronal sin depender de bibliotecas de aprendizaje automático, lo que permite entender a fondo los mecanismos de forward propagation y back propagation.

REFERENCES

[1] S. Srinivasan, "Audio MNIST Dataset," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/datasets/sripaadsrinivasan/audio-mnist/data. [Accessed: 30-Apr-2025].

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[3] M. Tan and Q. V. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in Proc. of the 36th International Conference on Machine Learning (ICML), 2019, pp. 6105-6114.

[4] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv preprint arXiv:1704.04861, 2017.

[5] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 2818-2826.

[6] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 4700-4708.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778.

[8] C. Tomasi and R. Manduchi, "Bilateral Filtering for Gray and Color Images," in Proc. of the Sixth International Conference on Computer Vision (ICCV), 1998, pp. 839-846.

[9] D. S. Park, W. Chan, Y. Zhang, C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition," in Proc. of Interspeech, 2019, pp. 2613-2617.

[10] Weights & Biases, "Experiment tracking and visualization for machine learning," [Online]. Available: https://wandb.ai/site. [Accessed: 30-Apr-2025].