Redes Neuronales Convolucionales y Backpropagation

Apuntes de clases

Rodolfo David Acuña López
Escuela de Ingeniería en Computación
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
rodolfoide69@estudiantec.cr

***Abstract*—En este documento podrá encontrar información sobre la semana 10 de clases de IA, donde se comparten las respuestas del quiz 5, se comentan detalles sobre el primer proyecto, un pequeño resumen de la clase anterior sobre back propagation y se habla sobre un tema nuevo donde podemos ver temas como ConvNet o arquitectura de red convolucional.**

***Index Terms*—Redes Neuronales Convolucionales, Backpropagation, CNN, Reconocimiento de Patrones, Procesamiento de Imágenes, Deep Learning**

I. Introducción

Las redes neuronales han revolucionado el campo de la inteligencia artificial, especialmente en tareas de reconocimiento de patrones y procesamiento de señales. En este documento se abordan dos conceptos fundamentales: el algoritmo de backpropagation, que permite el entrenamiento eficiente de redes neuronales profundas, y las redes neuronales convolucionales (CNN), que han demostrado ser particularmente efectivas para el procesamiento de imágenes y señales.

El backpropagation es un algoritmo de optimización que calcula los gradientes de la función de pérdida con respecto a los parámetros de la red, permitiendo su ajuste mediante descenso de gradiente. Por otro lado, las CNN introducen conceptos como convolución y pooling que aprovechan la estructura espacial de los datos, reduciendo significativamente el número de parámetros necesarios comparado con redes totalmente conectadas.

Este documento se estructura de la siguiente manera: primero se presentan aspectos administrativos del curso incluyendo respuestas del quiz y detalles del proyecto, luego se revisa el algoritmo de backpropagation con sus fundamentos matemáticos, y finalmente se introduce el concepto de redes convolucionales con la arquitectura AlexNet como ejemplo.

II. Aspectos administrativos

Debido a que no hubo noticias previas a la clase, se inició con una breve explicación sobre el primer proyecto de Redes Neuronales.

A. Respuesta del quiz

Se realizó el quiz 5 donde se establecieron las respuestas de este. Las preguntas con sus respuestas son las siguientes:

Pregunta: Describa qué es una red totalmente conectada (fully connected) **Respuesta:** Es un tipo de red neuronal en la que cada neurona está conectada con todas las neuronas de la capa siguiente.

Pregunta: Mencione 3 funciones de activación no-lineales. **Respuesta:** ReLU, Sigmoide y Tanh

Pregunta: Describa los 4 componentes principales de un agente LLM. **Respuesta:**

- **Perfil:** Puede tener su propia personalidad
- **Memoria:** Permite que el agente recuerde información pasada o resultados previos para mantener contexto en tareas largas
- **Herramientas:** Son funciones externas que el agente puede usar para ejecutar acciones
- **Planificación o razonamiento:** Decide qué hacer, interpretando las instrucciones del usuario y eligiendo la mejor acción

Pregunta: Describa la diferencia entre sistemas de agente único y sistemas multiagentes. **Respuesta:** Un agente único percibe su entorno, toma decisiones y actúa por sí mismo, mientras que los sistemas multiagentes son varios agentes que interactúan entre sí y con el entorno.

B. Explicación del proyecto

Para este proyecto necesitamos aplicar redes neuronales para el reconocimiento de voz a partir de espectrogramas, es decir, reconocimiento de patrones en voz donde utilizaremos una arquitectura que se llama Redes Neuronales Convolucionales (CNN) la cual nos sirve para el procesamiento de imágenes.

El reto de este proyecto es analizar una serie temporal en un audio donde analizaremos la señal que viene al segundo donde estemos procesando, por ejemplo, si estamos procesando un 5t, hay que procesar un 5t+1, 5t+2, y así sucesivamente. Se podrían resolver con redes recurrentes.

Otra forma de hacer esto es convertir esa voz en espectrogramas, la cual es un diagrama de tiempo y las frecuencias que produce la señal de audio. Con esto se produce un patrón.

Una herramienta mencionada por el profesor es Weights and Biases, la cual es una herramienta de seguimiento y visualización de experimentos de Machine Learning donde nosotros ejecutamos un entrenamiento y vemos en tiempo real desde cualquier computador cómo se está comportando un modelo. La ventaja es que podemos ver el comportamiento por lo que podemos detenerlo si no vemos buenos resultados.

El procesamiento de imágenes puede ser algo pesado por lo que debemos reducir el tamaño de estas a un tamaño de **224x224**. Esto debido a que computacionalmente se vuelve costoso.

No se pueden utilizar librerías como **ResNet** que sirven para abstraer la definición de capas más allá de torch.nn.

El profesor nos recomienda buscar una herramienta en redes neuronales que nos pueda hacer toda la arquitectura del modelo. Incluso esta se puede hacer en PyTorch, queda a nuestra disposición.

Tenemos dos modelos:

- **LeNet-5 clásico:** Este es la arquitectura más básica (como el profesor lo menciona) para el procesamiento de imágenes el cual fue creado por Yann LeCun.
- **Arquitectura alternativa:** Esta está basada en literatura la cual implementa cualquier arquitectura distinta.

Podemos escoger diferentes espectrogramas como por ejemplo, el Data Augmentation el cual trata de aumentar los datos de entrenamiento para mejorar la generalización de mi modelo con la finalidad de obtener mejores patrones.

En el paper SpecAugment, que sale en la bibliografía del proyecto, propone 3 tipos de técnicas:

- **Time Masking:** Donde tomo una frecuencia del 1 al 1.5 donde hago una máscara para cancelar el ruido
- **Time Warping:** Para estirar o encoger
- **Frequency Masking:** Que aplica máscaras similares pero en el eje de la frecuencia, lo que simula la pérdida o interferencia de ciertas bandas del espectro de audio

El entrenamiento se debe realizar varias veces por lo que se debe dejar todo montado y conectado la herramienta de Weights and Biases. Esto porque el entrenamiento con imágenes puede ser pesado, requiere de GPU. El profesor menciona que podemos usar Google Colab pero que es limitado, por lo que debemos aprovechar los recursos.

La extensión de la documentación debe ser de máximo 10 páginas. Los apuntes anteriores son los más relevantes sobre la explicación.

C. Repaso de Back Propagation

Este nos permite determinar cuánto aporta cada peso al error total de la red, ajustando los parámetros en la dirección contraria al proceso de propagación hacia adelante, así como lo podemos observar en la Fig. 1.

Vamos a ver las operaciones como grafos donde van a ser un tipo de operaciones donde vamos a ponerle un sobrenombre. El sobrenombre nos puede ayudar con las derivadas parciales. Cuando tratamos de optimizar un grafo, contamos con dos etapas. La de salida la cual le llamamos activación L donde

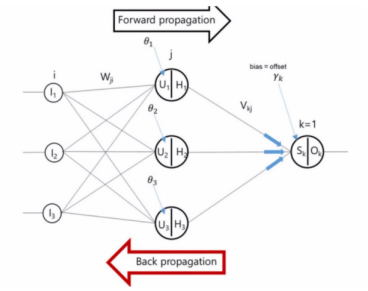

Fig. 1. Back and forward propagation.

tenemos solo una neurona. Cada una de esas neuronas están compuestas por una función no lineal que tiene como entrada una función lineal. Para optimizar los pesos en esa función debemos hacer derivadas parciales. Al final esa derivada va a ser el activador de la capa anterior por lo que no necesito conocer cómo fue computada cada capa anterior. Solo necesito el resultado y ya con eso puedo calcular la derivada que yo voy a necesitar. Si yo ocupo calcular la derivada parcial, respecto a la función de pérdida con mi parámetro w, lo que tengo que hacer es aplicar la regla de la cadena para llegar al parámetro de mi función. Hay cálculos que siempre se van a repetir por lo que podremos guardar esos cálculos para evitar recalcularlos de nuevo para cada uno de los parámetros.

$$\frac{\partial L_i}{\partial w^l} = \frac{\partial z^l}{\partial w^l}\frac{\partial a^l}{\partial z^l}\frac{\partial L_i}{\partial a^l},$$
$$\frac{\partial L_i}{\partial b^l} = \frac{\partial z^l}{\partial b^l}\frac{\partial a^l}{\partial z^l}\frac{\partial L_i}{\partial a^l}.$$

Esto nos da como resultado un vector gradiente, la cual podemos ver en la Fig. 2, que tiene el cálculo de todos los gradientes por todos los parámetros en la red.

$$\nabla L = \begin{bmatrix} \frac{\partial(L)}{\partial w^{(1)}} \\ \frac{\partial(L)}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial(L)}{\partial w^{(n)}} \\ \frac{\partial(L)}{\partial b^{(n)}} \end{bmatrix}$$

Fig. 2. Vector gradiente.

Si tenemos múltiples neuronas, tenemos que utilizar superíndices o subíndices, podemos ver un ejemplo en la Fig. 3. El primero me indica la capa en la que me encuentro, en este caso el L. El segundo me indica cuál neurona es para identificar cada una de ellas. Los pesos están asociados a las capas me van a indicar hacia dónde voy y de dónde provengo.

Podemos tener dos funciones, las cuales son:

- **Preactivación**

$$z_j^{(l)} = b_j^{(l)} + \sum_{k=1}^{n_{l-1}} w_{j,k}^{(l)}\, a_k^{(l-1)}$$

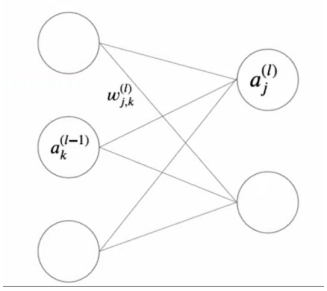

Fig. 3. Grafo dimensional.

- **Activación**

$$a_j^{(l)} = g\left(z_j^{(l)}\right)$$

La función de activación se aplica a toda la capa. Toda la capa se computa con sigmoide. Tenemos funciones de pérdida, donde podemos tener una pérdida total dada por:

$$L_i = \sum_{j=1}^{n_l} \left(a_j^{(l)} - y_j\right)^2$$

El resultado de la derivada de pérdida con respecto a la activación es la siguiente:

$$\frac{\partial L_i}{\partial a_j^l} = \left((a_1^l - y_1)^2 + (a_2^l - y_2)^2 + \cdots + (a_n^l - y_n)^2\right)'$$

$$\frac{\partial L_i}{\partial a_j^l} = 2\left(a_j^l - y_j\right)$$

El resultado de la derivada de activación con respecto a la de reactivación es la siguiente:

$$\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = g\left(z_j^{(l)}\right)\ \left(1 - g\left(z_j^{(l)}\right)\right)$$

En la siguiente figura yo puedo hacer varios cálculos de derivadas, donde a mí no me interesa cómo llegó el valor z ya que al final es un valor que me llegó a la función, donde se aplicaron varias derivadas para llegar a un valor. Al final, cada neurona que compute, no me interesa cómo me llegó la información desde la función de pérdida ya que a partir de cierto punto. Con el valor resultante, puedo sacar la derivada con respecto a x y con respecto a y almacenadas, y tener las derivadas desde mi función de pérdida con mis entradas. En resumen, la propagación hacia adelante es la capa de entrada por la de salida y la propagación hacia atrás es desde la capa de salida hacia la de entrada donde se calcula la gradiente del error con respecto a los pesos de cada capa.

III. ConvNet

Hasta ahora hemos trabajado con una red *fully connected* con entradas y salidas. El problema es que en el modelo anterior usado para **MNIST**, nos puede dar varios errores como si muevo las imágenes de un centro, y las neuronas que se activaban estaban fijas, entonces podemos tener un margen de error mayor. Básicamente las neuronas se desconectan de sus entradas originales y reciben otras para las cuales no fueron entrenadas.

Hay un dataset que se llama CIFAR-10 el cual son 10 clases con tamaño pequeño 32x32 pero son a color, con 3 canales. Por tanto, si tuviera que hacer una red neuronal para conectar cada píxel a una neurona, tendré una entrada de 3072 pesos. Con esto entramos a un problema de dimensionalidad. Esto se ve bien pero las imágenes son pequeñas, ¿qué pasa si se vuelven más grandes? Para resolver este problema, entra en juego el ConvNet, donde vamos a tener 3 dimensiones, donde vamos a tener filtros. Cada filtro se encargará de reconocer patrones en una imagen. Esos filtros pueden ser reconocimientos de líneas verticales, horizontales, diagonales, entre otras, que se especializan en extraer información de esas imágenes.

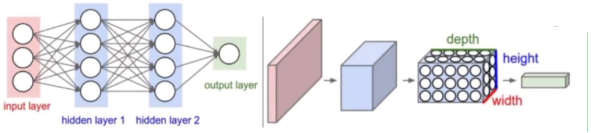

Fig. 4. Red totalmente conectada vs ConvNet

Mi salida se va a reducir, es decir, si tenía 224x224, mi salida puede ser 212x212, y si eran 3 canales, puede que ahora tengan 64 canales. Esa cantidad de canales van a representar la cantidad de filtros que yo tuve que calcular. Esos filtros pueden representar colores, líneas, números, entre otros.

La **arquitectura** está compuesta por 3 etapas:

A. *Convolutional Layer*

En esta se toma el filtro, se desliza por la imagen para hacer el cálculo de los features. Tenemos como entrada el largo, ancho y canales que voy a procesar. Esta computa la salida de neurona que se encuentran conectadas a las regiones locales. Por lo tanto, si se quiere usar una cantidad de x filtros, la salida de esta va a ser el ancho, largo y x. Después de los filtros se aplica una función de activación. Esos filtros se calculan por cada canal.

B. *Pooling layer*

Esta se encarga de reducir las dimensiones en largo y ancho, en otras palabras, reducir la imagen. Esta aplica la operación de downsampling a lo largo de dimensiones espaciales como el ancho y largo. Si su entrada es de 32x32x12, su salida puede llegar a ser de 16x16x16. Este no tiene parámetros y no me afecta la profundidad.

C. *Fully-connected*

En esta es más fácil reducir la información de una imagen a un vector más pequeño que el que yo tenía en la entrada, entonces, a partir de ese momento hago mi clasificador. En otras palabras, se calcula la probabilidad de que pertenezca a una clase y así convertir una imagen de píxeles a una probabilidad de pertenecer a una clase.

D. AlexNet

Esta es una arquitectura que salió para la revolución de convoluciones y el Deep Learning en todos sus aspectos.

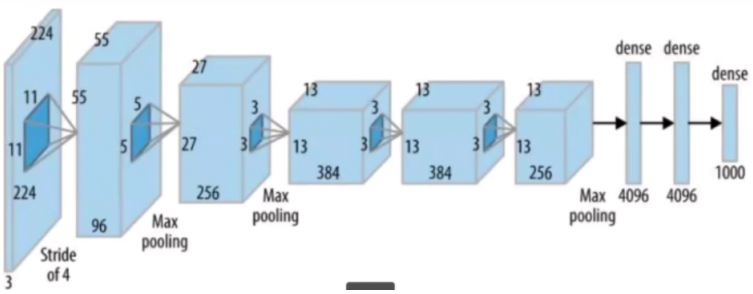

Fig. 5. Arquitectura AlexNet.

En la Fig. 5 podemos apreciar el proceso. Los cubitos de adentro representan el tamaño de los filtros.

De esta manera podemos tratar un problema de clasificación en imágenes. Lo primero que se hace es una convolución donde extraemos ciertas características. Para reducir las imágenes se realiza un pooling, donde se pierden píxeles. Al final llegamos a un resumen de la imagen anterior. Solo quedaría hacer una fully connected.

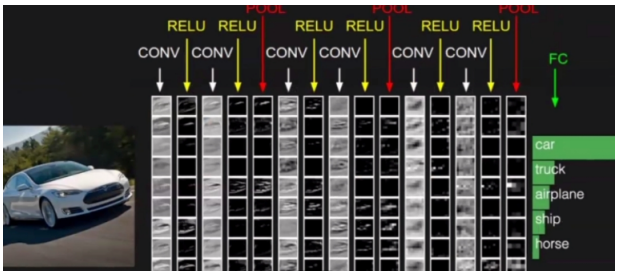

Fig. 6. Reducción de imagen.