UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN

Informe: Tarea #2

Informe sobre arquitecturas modernas de Redes Neuronales profundas
Universidad Central de Venezuela por el
Estudiante. Rafael Eduardo Contreras.

Profesor: Fernando Crema

Caracas, jul 15 de 2025

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN

ÍNDICE GENERAL

Notebooks

Notebook 10.1 - Convolución 1D - Link

Este notebook enseña las bases de la convolución de forma simple (en 1 dimensión).

Para implementar la función "**conv_3_1_1_zp**" investigue una función para agregar padding a un arreglo de numpy y luego aplique la convolución de forma estándar con np.sum sobre omega y el slice del arreglo donde corresponde.

Luego preguntan sobre porque los últimos outputs (de los bordes) son menores al original. Es simplemente porque uno de los valores agregados es omega_x * 0, ya que agregamos el valor del padding. El resto de omegas no es suficientemente grande para incrementar el valor por encima del original.

La siguiente convolución "**conv_3_2_1_zp**" es de la figura 10.3a-b. Esta tiene un stride de 2. Use la misma función de padding de antes y la diferencia es que ahora saltamos un espacio en el for loop, y el índice de la salida cambia a i//2.

La convolución "**conv_5_1_1_zp**" es similar a la "**conv_3_1_1_zp**" pero al tomar los slices del input incrementamos su tamaño para hacer match con el omega de tamaño. Finalmente, la convolución "**conv_3_1_2_zp**" fue la más difícil, debido al dilation de 2. Para esto escribimos la función de una manera menos genérica que las anteriores (Cambiar el dilation cambia la implementación fundamentalmente). Solo hicimos la multiplicación del omega con el elemento i, i+2 e i+4 del loop, tomando en cuenta que el input tiene suficiente padding.

La última implementación ("**get_conv_mat_3_1_1_zp**") consiste en crear la matriz de convolución. En este caso observar la figura del libro ayudo a resolverlo. No usamos padding, así que usamos condicionales para insertar los elementos de omega 0 y 2. De resto omega 1 siempre va en las diagonales. Omega 0 y 2 va i-1 e i+1 de la diagonal.

Una última pregunta que realizan es como aplicar 2 convoluciones sobre 1 input haciendo solo 1 operación de convolución (básicamente combinar convoluciones). Primero intenté hacerlo por mi cuenta descomponiendo la función de convolución (valor de h8[1] en el notebook) y tratando de deducir la fórmula. Luego hice algunos diagramas que me aproximaron al resultado final (el kernel de la convolución combinada). Finalmente por cuestiones de tiempo investigué como lograrlo y la respuesta es que se necesita aplicar la operación de convolución del primer kernel sobre el segundo, ya que la convolución es asociativa al aplicar este resultado al input se obtiene lo mismo que al haber aplicado ambos kernels por separado.

Notebook 10.2 - Convolución para MNIST-1D - Link

La primera sección de este notebook fue bastante sencilla, solo definir una red cuyas especificaciones son dadas en los comentarios. Es una red convolución de clasificación bastante simple con 10 clases (MNIST). Este modelo es entrenado y evaluado luego. El notebook nos muestra un gráfico con el error por epoch en entrenamiento y validación.

Notebook 10.3 - Convolución 2D - Link

En este notebook aplicamos operaciones de convolución 2d. El primer problema fue sencillo de resolver, ya que solo nos pedían obtener la imagen y peso a multiplicar para la salida. Solo había que prestar atención a los índices y tomar en cuenta que existía padding en las imágenes, básicamente manipulación de matrices.

En la segunda función **conv_numpy_2** nos agregan stride. Aquí el cálculo es similar a lo que hicimos en las convoluciones 1D, solo que para matrices (se multiplica o dividen los índices por el stride en ambas dimensiones).

Para la tercera función **conv_numpy_3** agregaron múltiples canales de entrada y salida. Esto fue sencillo, ya que las dimensiones ya eran correctas, solo que antes se hacía con 1 entrada y salida. Ahora solo es necesario agregar estos índices donde corresponden (channel_in para imagen, channel_out/in para weights y channel out para salida).

Finalmente, en **conv_numpy_4** es una convolución con múltiples imágenes. Similar a la parte anterior, solo hay que agregar el índice del batch.

Notebook 10.4 - Downsampling & upsampling - Link

Para la función **subsample** la implementación fue sencilla. Simplemente, copiar los elementos de la esquina. Para **maxpool** y **meanpool** el proceso fue casi el mismo, obtener regiones de números, aplicarles una función (max, mean) y usar este valor para llenar la casilla resultado correspondiente a la región.

Las funciones de upsampling fueron más complicadas. **Duplicate** fue sencilla (simplemente copiar). **Max_unpool** es regresar los valores máximos a sus posiciones originales, dejando el resto en 0. La función de **bilinear** fue la más complicada de implementar, al final decidí hacer un mapeo de los índices de los vecinos relativo a la salida. Luego dividí estos entre 2 para obtener los valores de la entrada (ya que es el doble de pequeña). Finalmente, obtuve el promedio de los vecinos de la entrada con estos índices y lo agregué a la salida correspondiente.

Notebook 10.5 - Convolution for MNIST - Link

Simple clasificación para MNIST. En este nos piden crear la red neuronal dadas sus especificaciones. Dos capas compuestas por una convolución, maxpoling y relu. Luego de esto las capas de clasificación lineales. Luego de esto nos muestran el entrenamiento y la precisión de predicción.

Experimento final

Realizamos un modelo de clasificación de imágenes. Usando imágenes de MNIST 100, donde clasificaremos números escritos del 00 al 99. Escogimos este dataset porque es similar al usado en el notebook 10.5, además de que este problema era mucho más sencillo de resolver comparado con problemas de segmentación de imágenes (más adelante). Asimismo, al momento de desarrollar el experimento se tenía una capacidad de cómputo media (gpu local, 4gb geforce 3050).

Hicimos leves modificaciones al modelo original para ver como se comporta. Hicimos dos nuevas versiones donde agregamos más capas de convolución con max pooling con una cantidad de canales que reduce la cantidad de parámetros.

Debido a la cantidad de datos y el tiempo de entrenamiento (durante el desarrollo del experimento usamos gpu local), se modificó el dataset para usar una fracción del mismo. A su vez, solo realizamos 5 batches para los entrenamientos.

El modelo original ya tenía buen performance y precisión. El objetivo de los siguientes modelos era reducir su tamaño y obtener los mismos resultados o mejores. El original tenía 851430 parámetros, la versión 1 tenía **318470** y la 2 **174550**.

Para reducir los parámetros se agregaban más convoluciones y capas de maxpooling. Esto reducía el tamaño del input para la siguiente capa, a su vez la cantidad de canales se incrementó de tal manera que la cantidad de parámetros siguiera siendo menor.

La versión 1 nueva con menos parámetros tuvo mejor performance que la versión original. Hay que tomar en cuenta que estos resultados vienen dados con una fracción de los datos y durante 5 épocas. La versión 2 del modelo con 174k parámetros tuvo un declive bastante significativo en la precisión, prácticamente el modelo dejaba de funcionar. Sobre la velocidad de aprendizaje, la perdida bajo más rápido en el modelo original que en el v1. Aun así, la precisión incremento más rápido en el v1.

Se logró mantener/incrementar la calidad del modelo original, reduciendo el número de parámetros entrañables.