Redes Neuronales 2020 – Trabajo Integrador

Universidad Nacional de Córdoba

Alumno: Simonian, Simón.

Profesor: Tamarit, A. Francisco

Diciembre de 2020

Introducción

Casi todos los días vemos en acción algún tipo de algoritmo de reconocimiento de objetos, por ejemplo, la detección facial de la cámara del teléfono móvil. La pregunta que la mayoría de las personas se hacen ante esto es, ¿cómo lo hace? En el núcleo de soluciones de visión computacional como éstas, se encuentran las redes neurales convolucionales (CNN).

Las neuronas de este tipo de red tienen un comportamiento muy similar a las neuronas de la corteza visual primaria de un cerebro biológico. Este tipo de red es muy efectiva para tareas de visión por compturadora como es la detección de objetos ó análisis de videos, como así también en la clasificación y segmentación de imágenes.

Las CNN contienen varias hidden layers, donde las primeras puedan detectar líneas, curvas y así se van especializando hasta poder reconocer formas complejas como un rostro, siluetas, etc.

Este trabajo tiene por objetivo realizar una red neuronal convolucional auto-encoder que logre replicar en el output las imágenes del dataset MNIST y comparar los resultados con la red neuronal Feed Forward auto-encoder del práctico 3. El autoencoder es un algoritmo de aprendizaje que permite el preprocesamiento de los datos con el objetivo que la red aprenda la función identidad, es decir que la entrada sea igual a la salida[1].

La entrada de la red es una imagen de 28 píxeles de alto y 28 píxeles de ancho, o sea 784 píxeles, en 256 tonos de grises. En la primer sección de este trabajo se dará una explicación más en profundidad de cómo funciona nuestra red creada, y en la segunda sección se llevará a cabo la comparación entre el funcionamiento de nuestra red y la red del práctico 3, usando como parámetros de comparación los errores de testeo y cómo replican la imagen de entrada cada una de las mismas.

[1] Cuando se parte de una red que ha sido autoaprendida con un auto-encoder el método del back propagation es más eficiente, ya que al realizar previamente este preprocesamiento de los acoplamiento se elimina el problema de la aleatoriedad de los pesos sinápticos iniciales.

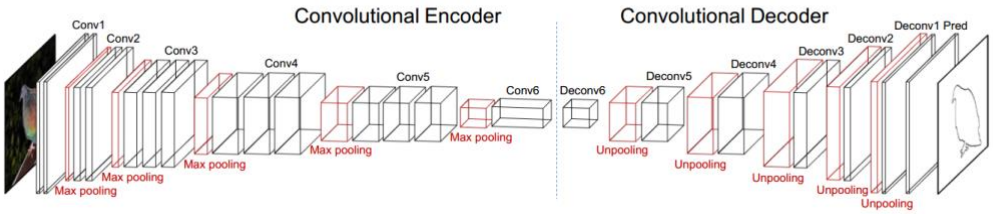
Sección 1: Funcionamiento de la red

Una vez importadas las librerías necesarias para correr la red (Torch, torchvision.transforms, torchvision, torch.nn, torch.nn.function, numpy, matplot, etc), cargado el dataset MNIST[2], dividiendo los valores de cada píxel por 255 para que los mismos varíen entre 0 y 1, y creados los dataloaders[3] de entrenamiento y testeo, el siguiente paso previo a definir la arquitectura de la red, es determinar el tamaño de los minibatch, el número de épocas, el optimizador y la función de error a utilizar:

- Se define un mibatch de tamaño 20.
- Se entrenará a la red durante 20 épocas
- Se trabajará utilizando como algoritmo de optimización ADAM, y se define un learning rate de 10^{-3}, betas de (0.9, 0.999), y un épsilon de 10^{-8}.
- Como función de error se utilizará la función Mean Square Error (MSELoss).

Ahora si se puede proceder a definir la arquitectura de la red.

Para explicar como funciona la arquitectura de la misma insertaremos una imagen que grafica y permite hacer más intuitivo el entendimiento de esta:



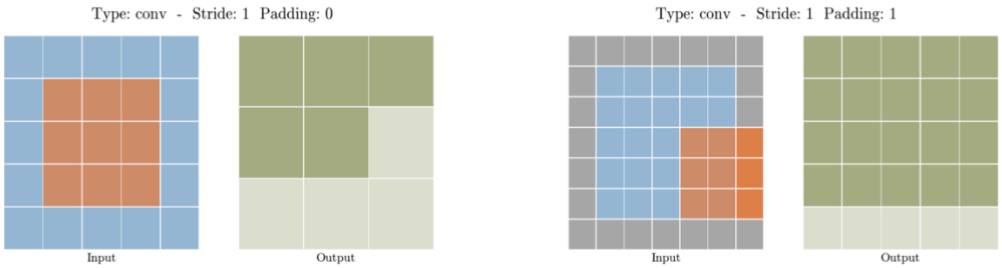
Convolutional Encoder:

En cada capa de la red debemos asignar la cantidad de kernels a aplicar, siendo los mismos los filtros que se se aplican a una imagen para extraer ciertas características importantes o patrones de esta. Matemáticamente lo que se hace para encontrar esos patrones es realizar la sumatoria de los productos entre los píxeles de la imagen con el valor del filtro en la posición del pixel de la misma. De esta cuenta se obtienen los distintos valores que asumen los píxeles de la nueva imagen. La función de activación usada es Relu.

En cada una de las capas lo que se aplica es una técnica denominada Max Pooling, como puede observarse en la figura anterior, método que ayuda a intensificar los patrones encontrados por los filtros aplicados en un paso previo y reduciendo la dimensionalidad de la imagen. Además en cada capa hay que definir los valores del padding y del stride y el orden del kernel. Para entender mejor a qué hacemos referencia con esto introducimos la siguiente imagen ilustrativa:

[2] Ya esta predeterminado que el dataset cuente con 60.000 imágenes dentro del conjunto de entrenamiento y 10.000 en el conjunto de testeo.

[3] Son iteradores creados sobre los conjuntos de datos de entrenamiento y test, que agrupan los datos en los batch y los mezclan si es necesario.



Donde el cuadrado azul es la imagen inicial, el cuadrado rojo representa el filtro aplicado (en este caso de orden 3) y lo verde es el output que la red arroja. Notemos que cuando el padding = 1, se agregan píxeles alrededor de la imagen original para que al realizar la convolución el output resultante sea de igual tamaño que la imagen original, siendo el valor de cada uno de esos píxeles cero, y el stride es un parametro indica cuantas posiciones ya sean columnas o filas se moverá el kernel. La fórmula para calcular el tamaño del output es la siguiente:

$$OutputSize = \frac{InputSize - Kernel + 2Padding}{Stride} + 1$$

Nuestra red en el encoder tiene 6 capas ocultas, 3 capas convolucionales y 3 capas pooling. El siguiente cuadro refleja la transformación del espacio a medida que trabaja cada una de las capas en la red.

Transformación del espacio	
Pasos a aplicar en el encoder convolucional	AltoXAnchoXProfundidad
Entrada	28x28x1
CNN1	28x28x32
MAXP1	14x14x32
CNN2	14x14x16
MAXP2	7x7x16
CNN3	7x7x8
MAXP3	3x3x8

La entrada es una imagen cualquiera de la base de datos MNIST y tiene por defecto el tamaño de 28x28x1 píxeles (en blanco y negro en nuestro caso, por eso el 1, ya que si fuera una imagen color tendriamos que multiplicarlo por 3). La primera capa es una convolucional y cantidad de filtros aplicados (me da la profundidad de la red) es de 32, con un kernel de tamaño 3x3, padding de 1 y stride igual a 1. El siguiente paso es

aplicar un Max Pooling de 2x2 y stride=2 sin padding para que la dimensión sea la mitad de la anterior, de forma que el output es de 14x14x32.

De igual manera se procede en los siguientes 2 pasos hasta llegar al espacio latente de 3x3x8. Cabe aclarar que en el último Max Pooling aplicado al ser el espacio de orden 7x7x8 (imparximpar), se ignora la última fila y la columna de más a la derecha.

Convolutional Decoder:

Luego, comienza la parte del Decoder. El método utilizado para llevar a cabo el decoder es usar una capa convolucional traspuesta, y mediante la misma se logra simplificar el paso de deconvolución y unpooling en una misma capa, es por esto que en nuestro caso para el decoder sólo tendremos 3 capas convolucionales traspuestas y una capa convolucional final que lleva como función de activación la Sigmoide.

Transformación del espacio	
Pasos a aplicar en el encoder convolucional	AltoXAnchoXProfundidad
CNNT1	7x7x8
CNNT2	14x14x16
CNNT3	28x28x32
CNNOUT	28x28x1

Además la fórmula para calcular el tamaño del output después de cada capa es:

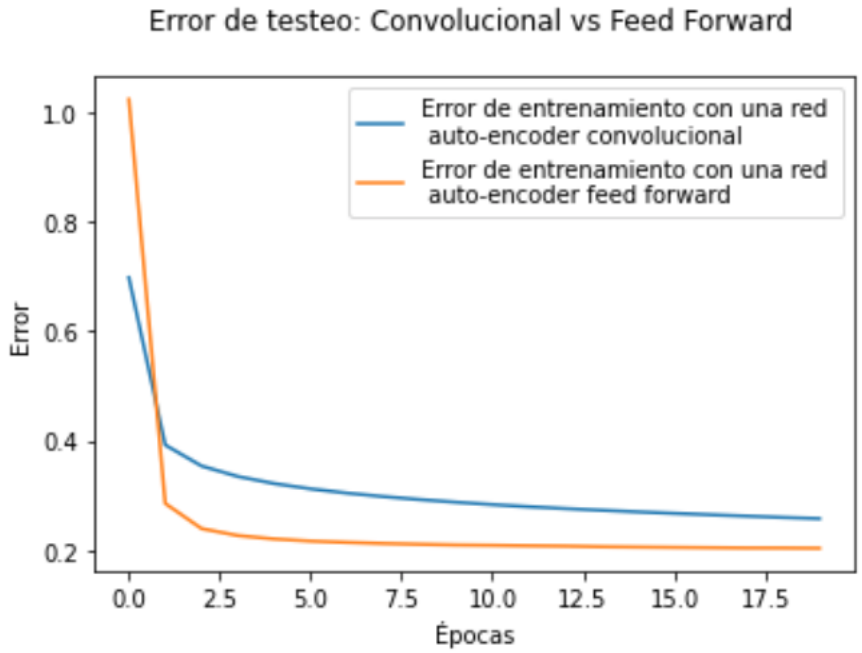
$$OutputSize = (InputSize - 1) * Stride + Kernel$$

Iniciamos con una capa convolucional traspuesta que nos arroja como output una dimensión de 7x7x8 (para ello se utiliza un kernel= 3, stride=2 y padding=0). En los siguientes dos pasos tenemos dos capas convolucionales traspuestas más, pero con un kernel y stride de 2 para duplicar las dimensiones. Se pasa de 7x7x8 a 14x14x16 y por último a 28x28x32.

La capa final es una convolución normal (k=3, s=2, p=1) que mantiene las dimensiones y reduce la profundidad a 1 tal que recuperamos la dimensión original de 28x28x1. Esta última capa como se aclaró antes lleva la función de activación Sigmoide.

Sección 2: Red Feed-forward Autoencoder vs Red Convolucional Autoencoder

Cuando se implementó una red feed-forward auto-encoder se trabajó con una capa oculta de 512 neuronas y una capa de salida de 784 neuronas, como algorítmo de optimización se usa ADAM con un learning rate de 10^{-3}, betas de (0.9, 0.999), y un épsilon de 10^{-8}, minibatch de tamaño 1000 y como función de pérdida se utiliza MSELoss.[4] Además se decide trabajar con 20 épocas y se utiliza el error de entrenamiento para llevar a cabo la comparación entre ambas redes. Los resultados obtenidos son los que muestra el gráfico a continuación:



Ambos errores tienden a caer rápidamente y a partir de la época 5 aproximadamente estabilizarse, aunque en el caso de la red autoencoder feed-forward la caída es más empinada y a partir de la época 10 la pendiente es prácticamente cero, estabilizandose en un valor del error cercano a los 0,2; mientras que para el caso de la red convolucional le pendiente es levemente negativa[5], y tiende a estabilizarse en torno a un valor del error de 0,24. Podemos notar que a pesar de que las redes convolucionales se utilizan específicamente cuando se trabaja con imágenes, hay una leve superioridad de la red feed-forward para llevar a cabo el aprendizaje de la identidad respecto de la red convolucional, pero esto puede darse porque el nivel de complejidad de la red convolucional es demasiado (se usan en total 10 capas ocultas) para un problema muy simple (es una imagen de dígitos numéricos entre 0 y 9, sólo con distintos tonos de grises).

A pesar de esto ambas redes terminan logrando un buen procesamiento de la identidad y logran replicar con muy buenos resultados a la imagen de entrada.

---

[4] Cabe aclarar que se trabaja con ADAM, MSELoss y 1000 de tamaño de minibatch ya que fueron los casos que consiguieron el mínimo error tanto de testeo como de entrenamiento con la red feed-forward auto-encoder para replicar la entrada.

[5] La variación del error de entrenamiento en la última época para la red feed-forward autoencoder es de $3*10^{-4}$, mientras que para la red convolucional autoencoder esta variación de la última época es de $3*10^{-3}$.

En la siguiente imagen se puede ver como la red convolucional auto-encoder replica algunas de las imágenes del dataset:

Donde las imágenes de la primer fila son las reales extraídas del dataset MNIST, y las de la fila de abajo son las que la red logra replicar.

Para el caso de la red feed-forward autoencoder tomamos dos imágenes puntuales y también logramos ver que la red lo replica de forma correcta: