Autoencoder Convolucional y Variacional

Nicolás Benjamín Ocampo
Licenciatura en Ciencias de la Computación - FaMAF
Redes Neuronales

Abstract—Este práctico aborda el estudio de una red neuronal convolucional encargada de estimar la función identidad. Procederemos a confeccionar dicha red aproximandonos a la solución óptima con una búsqueda de parámetros. Además se analizará el espacio latente de la misma en comparación de la red anteriormente trabajada y el producido por autoencoders variacionales.

Index Terms—Red Neuronal, Autoencoder convolucional, Autoencoder Variacional

1 Introducción

Ya vimos como una red neuronal nos permitió aproximar la función identidad "aprendiendo" los parámetros adecuados tratando el problema como uno de regresión. En dicha ocasión utilizamos una red feed-forward con una capa oculta obteniendo así una buena aproximación a la solución del problema pero con una gran cantidad de parámetros a ajustar.

Veremos una forma de reducirlos por medio de una red convolucional. Se aplicarán convoluciones o filtros entre sucesivas capas de la red para reducir la dimensionalidad del problema. Estos filtros van a captar la información importante con la que realizaremos el **encoder** de la red para luego proceder a realizar el proceso inverso de la convolución, es decir el **decoder**.

Por otro lado, nos interesará conocer el espacio latente de un autoencoder convolucional y los clusteres que se forman.

Por último veremos una variante del autencoder tradicional conocido como autoencoder variacional que nos permitirá mejorar el espacio latente.

Nuevamente utilizaremos la base de datos MNIST dada por un conjunto de dígitos escritos a mano y nos interesará obtener una imagen lo más idéntica posible a la entrada.

1.1 Topología de la Red

La Figura 1 muestra las topología de las red que vamos a utilizar. Notar que las funciones de activación entre capas estarán dada por una *ReLU* y que además no se usará *MaxPooling*. Para aumentar la dimensionalidad luego de alcanzar el bottleneck utilizaremos convoluciones traspuestas.

1.2 Parámetros de la Red

Los parámetros restantes a obtener son:

- La **loss** y su método de ajuste (SGD, Adam, etc).
- Learning rate η.
- Momento m (Si el método de ajuste lo requiere).
- Tamaño de batch y número de épocas.

Para elegirlos, tomaremos algunos parámetros que consideremos apropiados y procederemos a realizar las posibles combinaciones que se puedan dar entre ellos. En el caso de la **loss**, utilizaremos el error cuadratico médio (ECM) similar al caso de la red fully-connected que habíamos confeccionado.

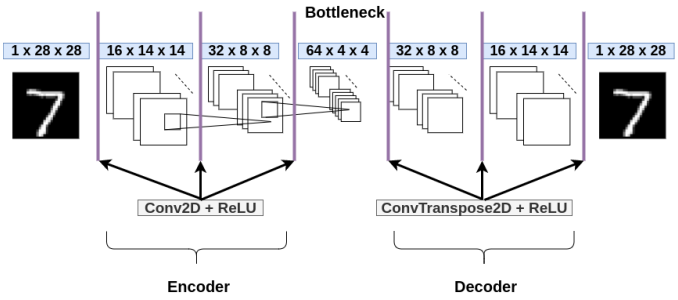

Fig. 1: Topología de la red convolucional.

Dichos conjuntos estarán dados por:

$$\begin{aligned} epocas &= \{8, 16\} \\ batch &= \{128, 256, 1000\} \\ M &= \{Adam, SGD\} \\ \eta &= \{10^{-1}, 10^{-2}, 10^{-3}\} \\ m &= \{0.2, 0.4, 0.6, 0.8\} \end{aligned}$$

Por lo tanto nuestras pruebas estarán dadas por el producto cartesiano de dichos conjuntos.

2 Resultados y Comparación

La Figura 2 muestra los resultados obtenidos por el autoencoder convolucional (AEC) en comparación al autoencoder con capas lineales (AEL).

Recordemos que el AEL estaba compuesto por una capa de entrada de 784 unidades, una capa oculta de 64 neuronas (utilizamos en este caso solo 64 y no las de 128, 256, o 512 en esta comparación, realizaremos variantes cuando analicemos el espacio latente) y una capa de salida de 784 neuronas. También contaba con dropout con $p = 0.1$ en la capa oculta pero se decidió removerlo así ambos modelos compartían los mismos parámetros. Estos fueron:

- Epocas = 16.
- Batch = 1000.
- Método Adam.

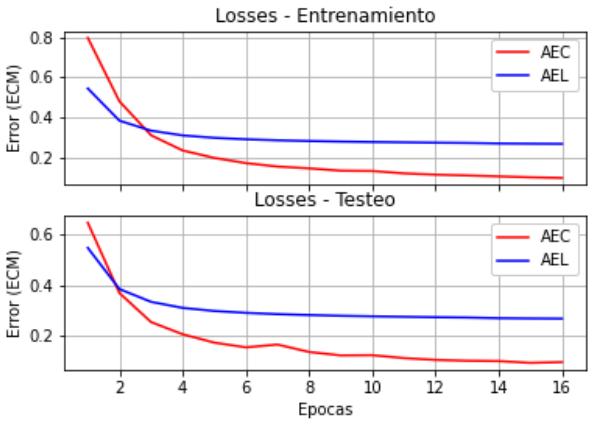

Fig. 2: Loss vs Epocas - Comparación de la loss de entrenamiento y de testeo entre el AEC y AEL.

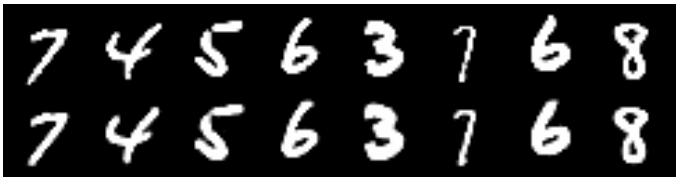

Fig. 3: AEC - Input (Arriba) **vs** Output (Abajo)

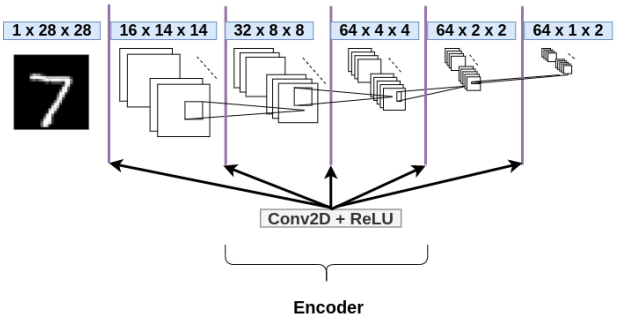

Fig. 4: Topología para reducir lo máximo posible el espacio latente del AEC

- Learning Rate $\eta = 10^{-3}$.

Se puede ver claramente que el aprendizaje es mucho más rápido por la convolucional según el gráfico **loss vs épocas** de la Figura 2. Notar además que el aprendizaje del AEL es casi nulo al trascurrir las épocas en comparación del AEC. La Figura 3 muestra algunas imágenes de entrada dada al AEC en comparación con su output de salida.

La lista completa de corridas efectuadas para el AEC puede encontrarse en (https://github.com/benjaminocampo/ConvolutionalAutoencoder_RUNS)

3 Espacio Latente

Ahora bien la capa más profunda en nuestra red convolucional es el espacio latente o encoded space, donde los datos de la capa inicial fueron comprimidos.

Si continuamos agregando más capas hasta obtener una de 1×2 podemos graficar dicho espacio en el plano. En nuestro caso utilizamos las capas dadas por la Figura 4. Si bien en el bottleneck muestra que se tiene 64 filtros de 1×2, vamos a encofarnos en uno de ellos. Cabe recalcar que se intentó reducirlo a tener un solo filtro, sin embargo al comprimir los datos a tal magnitud es necesario aumentar la cantidad de ellos en lugar de disminuirlos. Caso contrario el modelo no obtiene buenos resultados.

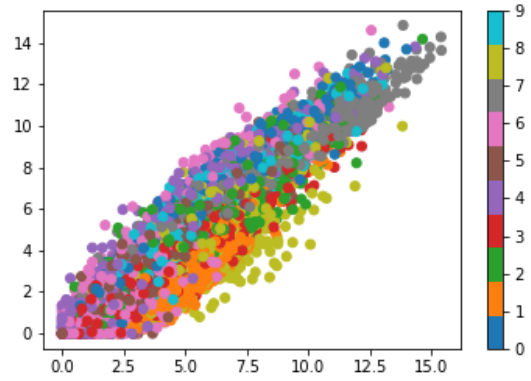

Fig. 5: Espacio Latente AEC

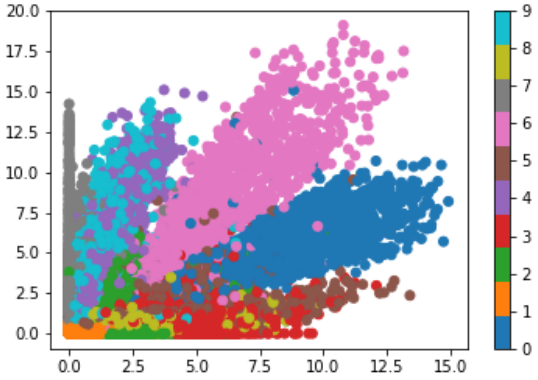

Fig. 6: Espacio Latente AEL

Por lo tanto, una vez entrenada la red y con un espacio latente de 2 dimensiones podemos codificar un batch de entrada usando su encoder y realizando un scatterplot de los vectores de salida.

La Figura 5 muestra dicho diagrama indicando con una etiqueta que dígito de entrada fue dado al encoder, y en que posición del espacio latente le corresponde. Algo a notar es que los puntos si bien forman pequeños clusteres varios de ellos están solapados no tan diferenciados entre sí, probablemente debido a que solo estamos contemplando uno de los filtros y no su totalidad.

Si confeccionamos el mismo gráfico con la red fully-connected obtenemos lo dado por la Figura 6.

En este caso, para reducir el espacio latente hasta 2 neuronas se vio la necesidad de cambiar la topología anterior a 3 capas ocultas. Una de 512 neuronas, otra de 2 y la última de 512 para mantener la simetría del autoencoder. Esto se debe a que nuevamente no se obtenían muy buenos resultados de otra manera.

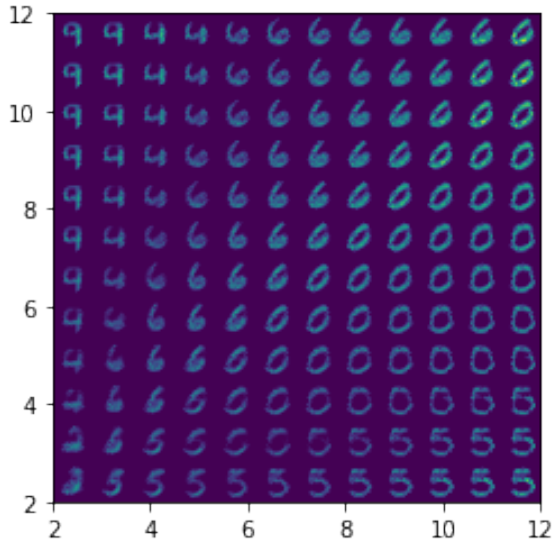

Fig. 7: Decodificación de muestras del espacio latente del AEL.

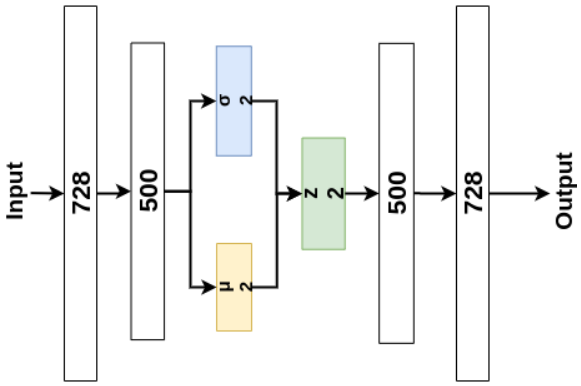

Fig. 8: Topología del Autoencoder Variacional.

Ahora bien, los clusteres de los vectores latentes de dígitos simliares resultaron estar en las mismas regiones y no tan solapados. Sin embargo entre ellos se pueden ver ”huecos” que corresponden a secciones del espacio latente que no fueron mapeadas. Por lo tanto si el decoder recibiera como dato de entrada un punto de esas secciones el mismo no sabría como interpretarlos.

Para verificar esto podemos generar de manera uniforme muestras del espacio latente y ver como el decoder reconstruye dichos inputs. La Figura 7 muestra las decodificaciones de los dígitos.

Notar que las mismas corresponden con la disposición del espacio latente como habíamos predicho. Por ejemplo hacia la derecha del gráfico se decodifica el dígito 0, que corresponde con los puntos azules de la Figura 6.

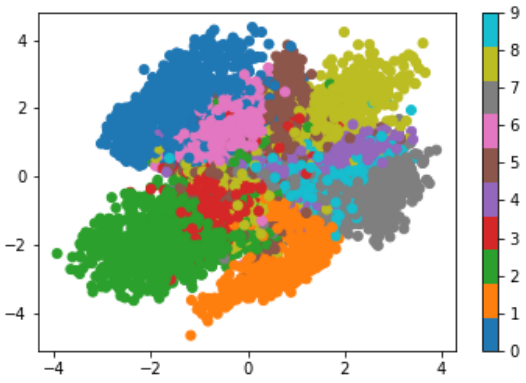

Fig. 9: Espacio Latente VAE.

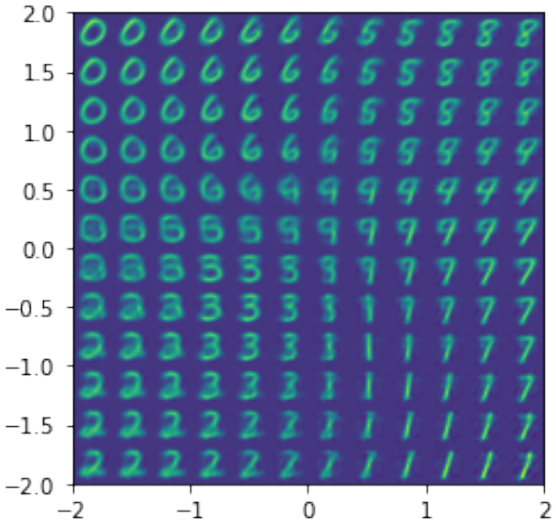

Fig. 10: Decodificación de muestras del espacio latente del VAE.

Ahora bien, parte de la muestra se posicionó sobre dichos ”huecos” y por lo tanto la decodificación fue menos clara.

Por simplicidad solo producimos las muestras para el autoencoder no convolucional (Dado que la cantidad de filtros es 64 en el espacio latente deberíamos generar muestras para cada filtro).

4 Autoencoder Generativo

Uno de los problemas del autoencoder tradicional es que el espacio latente tiende a dejar secciones de datos que nunca fueron mapeados. Por otro lado, en el caso convolucional a priori vimos que no se presenta esto pero nos gustaría que el espacio latente esté mas centrado y menos solapado.

Además, queremos que el decoder sea más robusto en el sentido de que permita generar imágenes de datos que no fueron exactamente mapeadas en un punto del espacio latente, pero que está lo suficientemente cerca de uno que si lo fue, es decir, que el mismo sea continuo en lugar de discreto como en los casos anteriores.

Para este caso, utilizaremos autoencoders generativos (VAEs) que en lugar de mapear un input x a un vector z del espacio latente, se lo hará hacia un vector meadia μ y desviación estandar σ donde luego obtendremos una muestra z del espacio latente, solo que dicha muestra tendrá distribución $N(\mu, \sigma)$.

La Figura 8 ejemplifica la confección de dicha red (Notar que nuevamente utilizaremos redes fully-connected). Por otro lado, se debe agregar una loss auxiliar conocida como KL que evita que tengamos puntos de un mismo dígito con media muy diferente y una pequeña desviación estandar.

En las Figuras 9 y 10 se pueden ver el espacio latente generado y la decodificación de muestras de dicho espacio.

Ahora podemos ver que su disposición es mucho más centrada, y sin separaciones entre ellos. También las decodificaciones fueron mucho más nítidas a diferencia de un autoencoder no generativo.

5 Conclusión

- Vimos como el aprendizaje de una red convolucional fue ligeramente superior al de una red con capas lineales debido a que estamos preservando la información de la disposición de la imagen. Cabe aclarar que si dichas imágenes fuesen de tamaño superior el uso de convolucionales sería más que necesario. Si se estuviera buscando catalogar los dígitos allí'sería más conveniente una lineal o combinaciones de ambas.
- Por otro lado, si se está buscando generar datos nuevos a partir de algunos iniciales, lo más adecuado es un autoencoder variacional cuya disposición del espacio latente es más suave y centrada y permite mayor flexibilidad en la decodificación.
- Los parametros finales de mejores resultados para el AEC fueron:
  - Epocas = 16.
  - Batch = 1000.
  - Método Adam.
  - Learning Rate $\eta = 10^{-3}$.

References

[1] J. Hertz, A. Krogh and R.G Palmer, *Introduction to the theory of neural computation*, Santa Fe Institute (1991).

[2] C. Soumith, *Deep Learning With Pytorch: A 60 Minute Blitz*, en https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html (2017)

[3] A. Van De Kleut, *Variational AutoEncoders (VAE) with PyTorch* en https://avandekleut.github.io/vae/ (2017)

[4] I. Shafkat, *Intuitively Understanding Variational Autoencoders* en https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf (2018)

[5] J. Rocca, *Understanding Variational Autoencoders (VAEs)* en https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 (2019)