Aprendizaje Automático con Python

Francisco Marsiglione

Octubre de 2020

---

1. Sobre el problema

Se propone implementar un autoencoder para imágenes en el formato MNIST[1] (en blanco y negro, de 28x28 pixeles). Las precondiciones serán el uso de la función de error cuadrático medio, un descenso por el gradiente estocástico, y *Dropout* con probabilidad $p = 0{,}1$. También se fija el tamaño del minibatch (tanto para entrenamiento como testeo) en 1000. Todos los demás parámetros serán variables o se ajustarán en algún punto del testeo; estos son:

- N° de neuronas en la capa oculta (al principio 64).
- Learning rate.
- Factor de momento (aunque no se aclara su necesidad, veremos que hace la diferencia a la hora del aprendizaje).
- Durante el entrenamiento, la cantidad de *épocas* que se dedicarán.

2. Desarrollo y resultados preliminares

En *Colab* y con el uso de la librería *PyTorch*[2], es simple construir una red neuronal de este tipo. Tomamos la base de datos MNIST de esa misma librería y la adaptamos al cálculo de la identidad. En términos de *PyTorch*, nuestra red tendrá una función que modificará el *tensor* de entrada (la imagen), paso por paso, de la siguiente forma:

[1] http://yann.lecun.com/exdb/mnist/
[2] https://pytorch.org/

1. Flatten: convertimos la imagen 28x28 en un arreglo de 784 pixeles.

2. Una capa lineal (que hace combinación lineal de los pesos) para la capa de entrada y otra para la salida (o visto de otra forma, una que actúa sobre la entrada y otra sobre la capa oculta).

3. Una capa que aplica la función de activación (en esta red usamos la Sigmoide) luego de cada capa lineal.

4. Una capa de Dropout ($p = 0{,}1$) luego de la capa oculta (más allá de la desactivación aleatoria de neuronas, no hace más que calcular la identidad, así que no debemos hacer más nada con esta capa).

5. Reconstruct: al final, recuperamos el formato de 28x28. Esto es útil para comparar rápida e intuitivamente la imagen original con la procesada por la red. De hecho, hay una razón por la que se eligió la función Sigmoide como activación: si *evitamos* normalizar los datos de MNIST, por defecto los pixeles en *PyTorch* son números del 0 al 1, igual que la imagen de la función Sigmoide, lo que produce resultados comparables por la fución de error. Más aun, podríamos reemplazar la función de costo o error por la *Cross Entropy*, también en la librería provista, y la red seguiría funcionando.

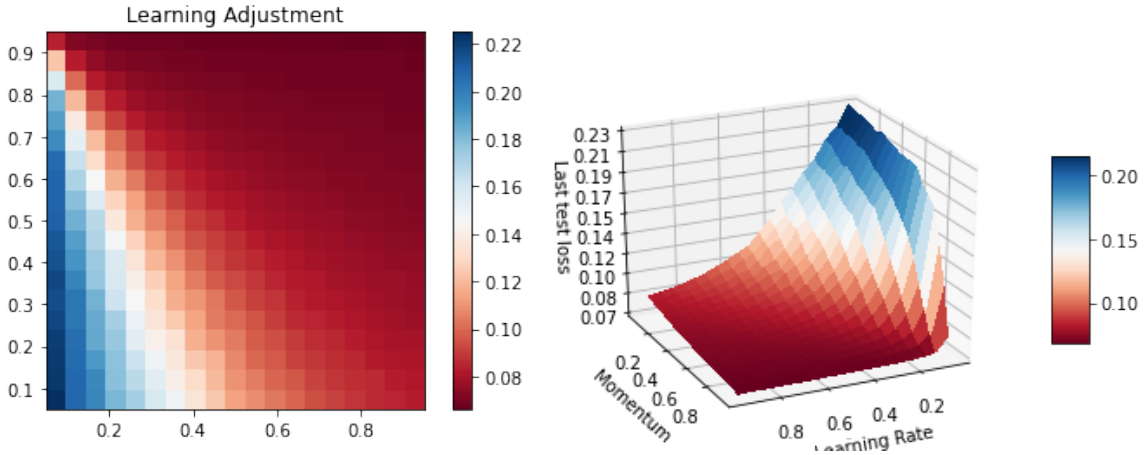
Los resultados son prometedores. Sin falta, la red asigna pesos iniciales (aleatorios) que producen un error cuadrático medio de entre 0,2 y 0,25, pero esto disminuye a $\sim 0{,}08$ en las primeras 10 épocas (para la mayoría de las configuraciones, excepto para valores bajos de los parámetros, como se verá en la siguiente sección). Con buenos parámetros, esto puede lograrse en las primeras $2-5$ épocas. En cuanto a cómo interpretar esto, notar que el comportamiento por defecto de la función de error usada es computar el promedio, por lo que el número obtenido es la distancia cuadrática media *por pixel* (siendo cada pixel un valor entre 0 y 1). Naturalmente, esto quiere decir que la distancia absoluta para $loss = L$ es $\sqrt{L}$. Al principio, estamos alejados en $0{,}45 - 0{,}5$ por pixel!

3. Análisis de parámetros

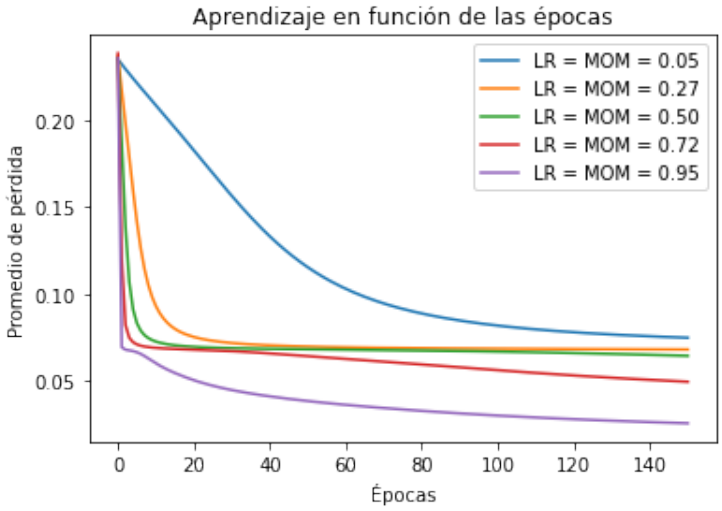
Quizás la primera pregunta que siempre deberíamos hacernos antes de poner en marcha una nueva red concierne a los parámetros. ¿Cuál es la configuración óptima para este problema en particular? Por mera inspección, se observa que tanto el *Learning rate* como el *Momento* afectan drásticamente la tasa de aprendizaje. Más aun, es aparente que logramos mejores resultados con valores altos de ambos (siempre en el rango $0 - 1$, como indica la convención). Pero, ¿Cuál es la combinación ideal?

Se aprovechó este problema para hacer una primera prueba de la red funcionando. Los primeros gráficos muestran el último error medio de testeo obtenido luego de sólo 5 épocas para 361 redes similares, pero combinando distintos valores de learning rate y momento (saltos de 0,05 entre 0,05 y 0,95, lo que nos da $19 * 19 = 361$ combinaciones). Esto equivale, claro a está, a un total de 1805 épocas.

Como se ve, efectivamente tenemos mejores resultados (menor error) con valores más altos de los parámetros. Más aún, al graficar como una función de 2 variables notamos que cada parámetro disminuye el costo independientemente del otro.



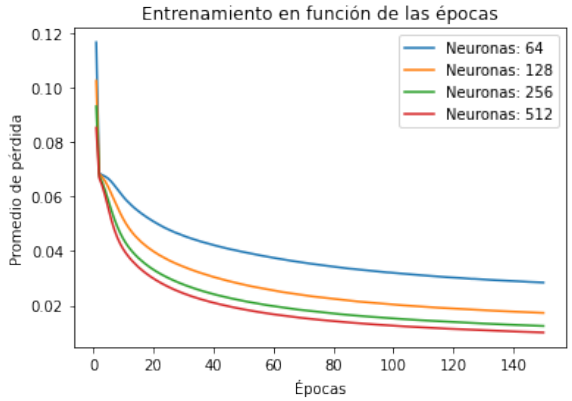
Pero, ¿Puede confiarse en este resultado al largo plazo? Un learning rate alto, por ejemplo, puede ser un problema luego de muchas épocas. Para esto, se hizo un análisis adicional con menos combinaciones de parámetros, pero un período de 150 épocas. Los resultados son bastante seguros; desde ahora, usaremos $LR = MOM = 0{,}95$ (aunque podríamos usar valores más altos).

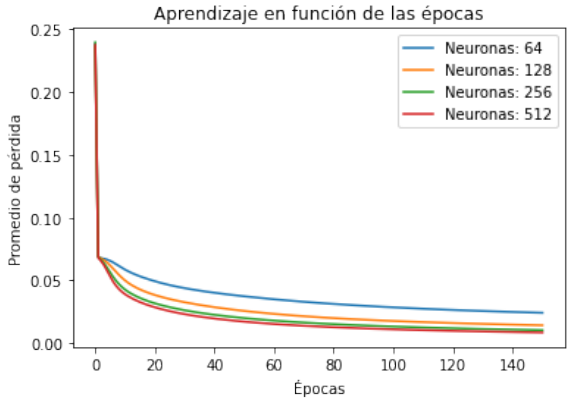


4. Análisis de la capa oculta

Habiendo fijado 2 de los 3 parámetros problemáticos, nos interesa ver qué tan eficiente es la red con distinta cantidad de neuronas en la única capa oculta. En particular, el análisis previo se hizo con 64 neuronas, y queremos comparar con 128, 256 y 512. Siguiendo el ejemplo anterior, ahora fijamos los parámetros de aprendizaje en 0,95 y vamos variando el número de neuronas, siempre con un período de 150 épocas. Aquí, "Entrenamiento" hace referencia al error producido durante las etapas de *training*, mientras que "Aprendizaje" se refiere a las etapas de testeo.

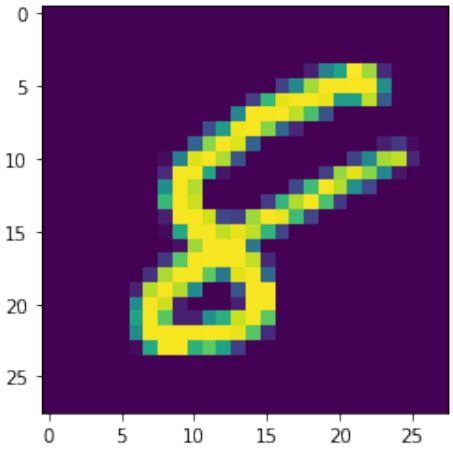
Como indica la intuición, un número alto de neuronas ayuda a un aprendizaje más veloz, aunque resulta cada vez más difícil mejorar el resultado cuando aumentamos ese número; en efecto, no hay una gran diferencia entre el aprendizaje con 512 y 256 neuronas, pero 64 neuronas son muy pocas.





5. Demostración práctica

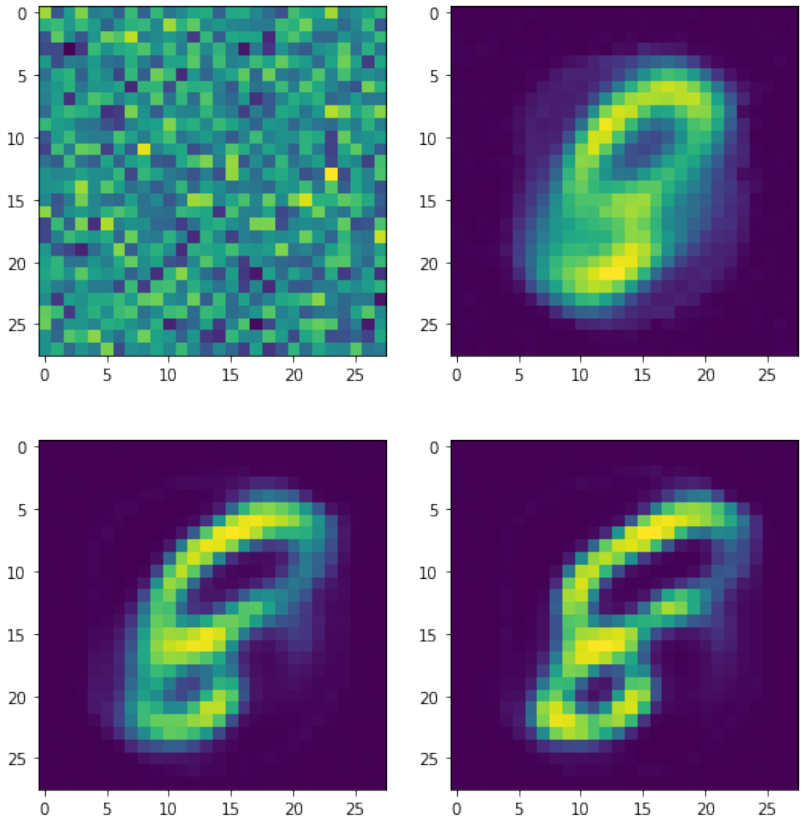
Ahora bien, nos interesa comprobar el desempeño real de la red. Aunque sería inusual, todos los resultados anteriores podrían ser causados por dos o más errores que se compensan (como una red mal construida con un error mal calculado). Normalmente, el autoencoder se usa para configurar los pesos iniciales de manera más conveniente para la red *real*, la cual podría ser un clasificador. Pero, por sí mismo, el autoencoder computa la identidad; en otras palabras, debería ser capaz de dibujar *por su cuenta* (sin "pasar" el resultado directamente) la imagen que le mostramos, y *sin haberla visto antes*. Para ver esto, elijamos alguna imagen del conjunto de testeo de MNIST; digamos, la 485. Esta se ve así:



El label correspondiente nos informa que es un 8, pero esto es irrelevante para esta red; la escencia del autoencoder es el aprendizaje *no supervisado*, porque el label no juega ningún papel el la correctitud del resultado. De hecho, no podemos decir qué resultado es correcto, sino solamente qué tan lejos está del resultado ideal (la imagen original). Esto se logra con el error de testeo. Notar que esta imagen nunca forma parte del entrenamiento de la red (no se hace *back-propagation* cuando se llama a la red con esta imagen).

A continuación se muestran los intentos de la red de dibujar esta imagen en distintos puntos de su entrenamiento: 0 épocas, 10 épocas, 50 épocas y 100 épocas. Naturalmente, antes de empezar el entrenamiento, la red no sabe cómo dibujar, pero se acerca bastante luego de sólo 10 épocas, como se vio anteriormente. Notar que mientras más aprende, más le cuesta completar los detalles, y necesitamos muchas más épocas. El hecho de que

100 épocas alcancen para que el ojo humano identifique la imagen de salida como la misma de entrada tiene que ver con que usamos todos los parámetros más convenientes determinados anteriormente: $LR = MOM = 0{,}95$ y 512 neuronas en la capa oculta.



El uso de la Sigmoide como función de activación y la reconstrucción de la imagen al final de la red facilita la comparación entre el input y el output; esto es, podemos calcular el error cuadrático medio por pixel, pero de los 784 pixeles de esta imagen solamente. En los resultados de arriba, los errores respectivos son 0,233811, 0,051340, 0,028528 y 0,018116. Si continuamos durante 500 épocas, obtenemos un error de 0,006251, correspondiente a la siguiente imagen: