Redes Neuronales
Trabajo Final
CNET

Igor Andruskiewitsch

2020

1 Introducción

En este trabajo vamos a ver la estructura y lógica detrás de construir una red neuronal artificial (*ANN*) desde cero con el objetivo de automatizar las clasificación de dígitos del dataset MNIST.

Vamos a analizar la arquitectura y los hiperparámetros utilizados, los resultados obtenidos y algunas particularidades de la implementación. No vamos a analizar el algoritmo de optimización ni su implementación, por cuestiones de simplicidad, aunque si vamos a explicar las singularidades y limitaciones que presentó la implementación.

1.1 Objetivos

- Explicar el diseño y la arquitectura
- Listar las limitaciones
- Describir la arquitectura de red utilizada para clasificar dígitos del *MNIST*
- Visualizar los resultados del punto anterior

2 Diseño

El diseño que elegimos para la implementación de nuestra red neuronal, se basa en dos componentes:

- **cnet** es un contenedor para nuestra red, contiene la información de *entrada/salida* y nuestras capas
- **clayer** representa cada una de las capas de nuestra red, almacena:
  - una matriz de *pesos*
  - un arreglo de *tendencias* (bias)
  - un tipo de *activación* (limitadas en la implementación)
  - un arreglo llamado *delta*, que contiene pasos intermedios del algoritmo de backprop
  - un arreglo *output*, que contiene el último vector que dió como resultado la capa

También definimos otros módulos, que nos permiten manipular con facilidad los datos resultantes o las funciones necesarias durante el entrenamiento de nuestro modelo.

- **cnet_loss_type** define los tipos de *pérdidas* soportadas, para las cuales se define una implementación y su derivada:
  - *mse_loss* (Error Medio Cuadrado)
  - *cross_entropy_loss* (Entropía Cruzada)
- **cnet_act_type** define los tipos de *activaciones* soportadas, para las cuales se define una implementación y su derivada:
  - *relu_act* (ReLU)
  - *sigmoid_act* (Sigmoide para clasificación binaria)
  - *softmax_act* (Softmax para clasificación con multiples clases)

3 Limitaciones

En esta sección vamos a desarrollar algunas de las limitaciones con las que contamos a la hora de desarrollar la librería

3.1 Algoritmo de backprop

El algoritmo elegido para actualizar los pesos de nuestra red es el **GD***(Gradiente Descendiente Simple)*. Esto significa que no vamos a utilizar batches (lotes) para inferir con nuestro modelo como se hace con el **SGD**(Gradiente Descendiente Estocástico). Esto se debe a la complejidad agregada de variar la forma del input a tomar por nuestro modelo, simplificamos esta variable utilizando un único input. Esta decisión trae consigo algunos puntos en contra, ya que es necesario dedicar más épocas de entrenamiento, aunque vemos que cada época representa una caída en la pérdida, a diferencia del *SGD*, que puede variar generando caídas o subidas en la pérdida en cada epoch.

3.2 Capas disponibles

Otra limitación a considerar son las capas disponibles, que en este caso son únicamente **capas neuronales**, es decir que no contamos con **dropout**, **convoluciones** ni otros tipos de capa. Esto significa que no podemos utilizar la librería para realizar análisis sobre imágenes complejas (por eso la utilización del *MNIST*). También es necesario tener cuidado con los hiperparámetros para evitar el *sobreajuste*, ya que no contamos con la capa de **dropout** para ajustar estos valores.

3.3 Hiperparámetros y optimizaciones

Para nuestra librería, el algoritmo de entrenamiento cuenta con parámetros limitados, sólo podemos proveer el *learning rate* y la cantidad de *épocas*. No contamos con un método para realizar *weight decay*, ni *momentum*. Estos métodos podrían generar un entrenamiento mucho más certero pero no se incluyeron para evitar acomplejizar la librería.

4 Clasificación de *MNIST*

En esta sección vamos a explicar la arquitectura que elegimos para un modelo que clasifique los dígitos del dataset *MNIST*. Esta es una tarea de clasificación de múltiples clases, ya que el dataset cuenta con dígitos entre 0 y 9 y queremos que nuestro modelo sea capaz de predecir, dada la imágen, a qué digito corresponde. A primera vista, la mejor arquitectura incluiría capas convolucionales, en conjunto con un output activado por la función *Softmax*, que limita la suma de las probabilidades

a 1 y una función de pérdida de *Entropía Cruzada*. En nuestro caso, vamos a utilizar la función de activación *Sigmoide*, que limita cada probabilidad a un número en 0 y 1, y como función de pérdida, vamos a tener la función *MSE*. Si bien no es la combinación ideal para la tarea, veremos que obtuvimos resultados bastante buenos.

La arquitectura capa por capa de la red es la siguiente:

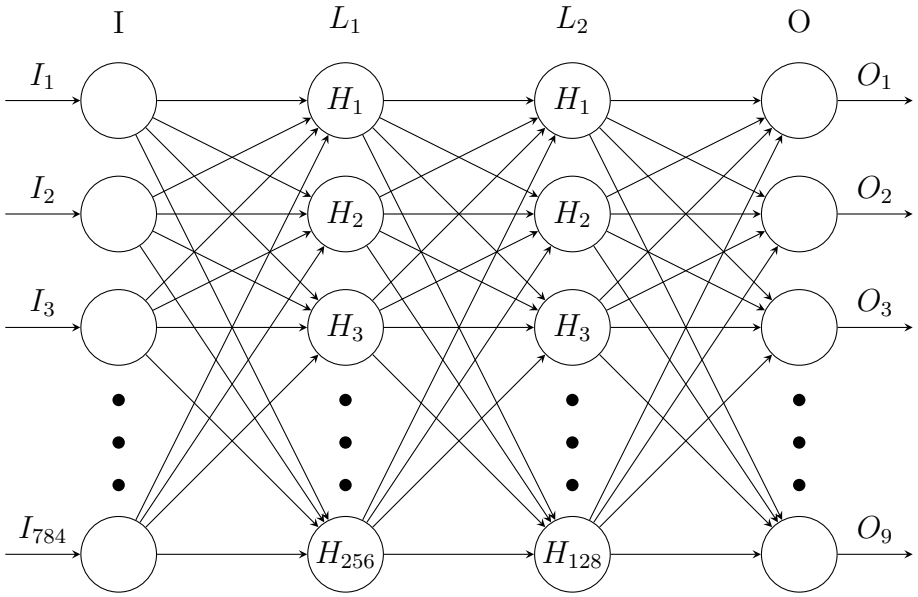


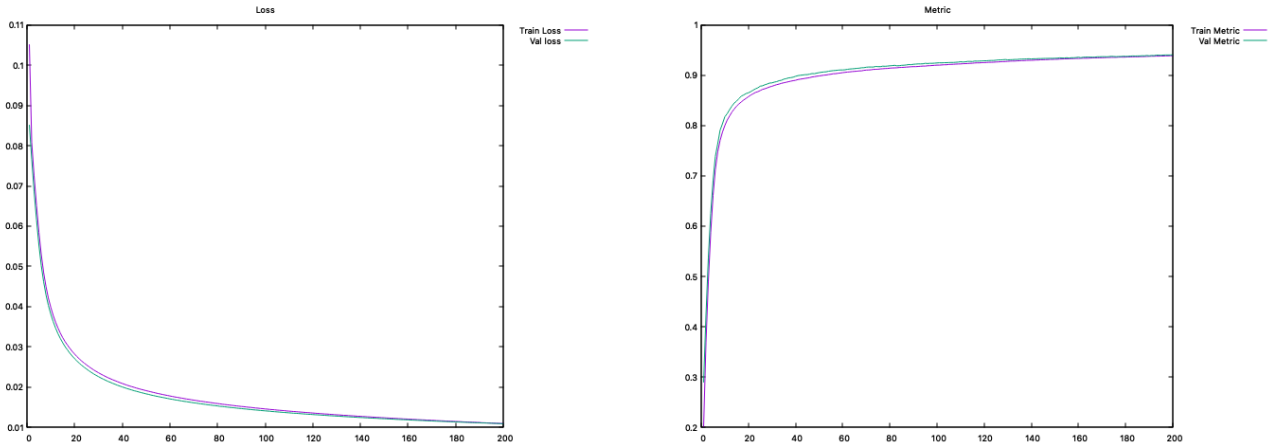
Figure 1: Arquitectura para nuestro modelo de clasificación.

Vemos que tenemos un input de tamaño 784, que las imágenes del *MNIST* son de $28x28$, tenemos dos capas ocultas de tamaño 256 y 128 respectivamente, y recibimos como salida un vector de tamaño 10, con las probabilidades correspondientes para la clasificación de los dígitos del 0 al 9.

5 Resultados *MNIST*

La arquitectura descripta en la sección anterior fue entrenada por 200 épocas. Este proceso tardó aproximadamente unas 5 horas.

Podemos observar en la figura 2 que las curvas de las métricas de los datos validación y entrenamiento no tienen grandes diferencias, por lo que podríamos pensar que no hubo sobreajuste. La pérdida y la exactitud finales del modelo fueron de aproximadamente 0.01 y 0.94 respectivamente.



(a) (b)

Figure 2: Métricas de entrenamiento

Una vez terminado el entrenamiento, también se realizaron tests sobre los datos de prueba. De estas pruebas, obtuvimos la siguiente matriz de confusión

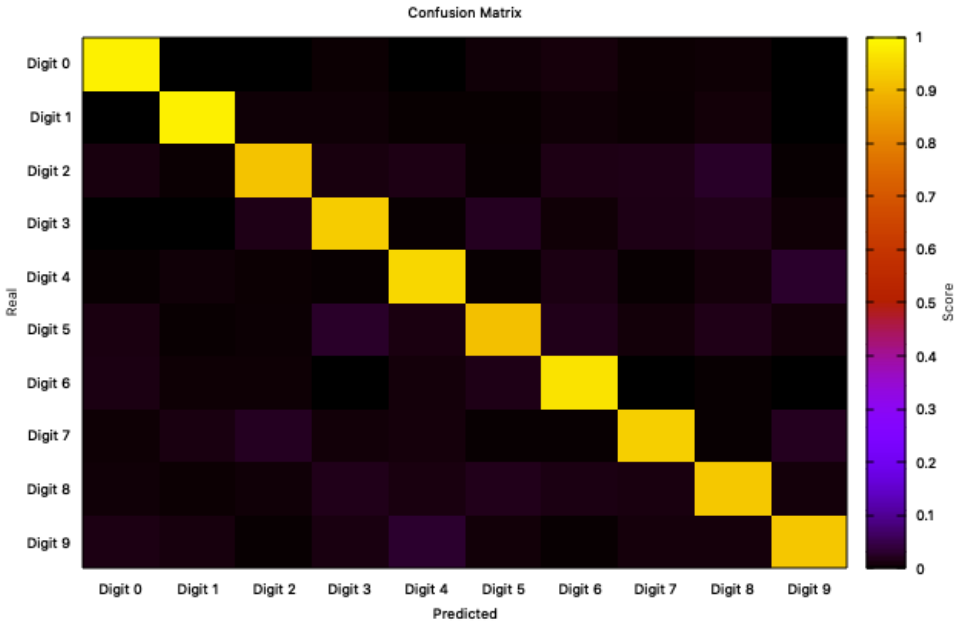


Figure 3: Matriz de confusión para los datos de validación del MNIST

En esta podemos observar que la gran mayoría de los dígitos fueron clasificados correctamente, aunque hubo algunos dígitos clasificados incorrectamente. Para comprender mejor cuáles son los dígitos que generan problemas en nuestro modelo, generamos un reporte de clasificación, que incluye distintas métricas:

	precision	recall	f1-score	support
Digit 0	0.951533	0.981633	0.966349	980
Digit 1	0.973753	0.980617	0.977173	1135
Digit 2	0.948744	0.914729	0.931426	1032
Digit 3	0.931480	0.928713	0.930094	1010
Digit 4	0.925224	0.945010	0.935013	982
Digit 5	0.928981	0.909193	0.918980	892
Digit 6	0.933131	0.961378	0.947044	958
Digit 7	0.947628	0.932879	0.940196	1028
Digit 8	0.919140	0.921971	0.920554	974
Digit 9	0.936492	0.920714	0.928536	1009

Final Accuracy: 0.940300 - Samples: 10000

En este reporte podemos observar que, a excepción del 0 y el 1, la mayoría de los dígitos tienen métricas menores a 9.5, esto puede deberse a que el 0 y el 1 son más dígitos comunes en dataset (la métrica **support** del reporte es de los datos de validación y no de entrenamiento). Aún así las métricas son buenas, no bajan de 0.9 y podrían mejorarse utilizando técnicas no soportadas mencionadas anteriormente.