Fashion-MNIST con PyTorch

Kevin Gaston Mansilla*

(Dated: November 18, 2024)

I. INTRODUCCIÓN

Las redes feed forward son un tipo de red neuronal que estan compuestas por N capas donde $N-1$ son capas ocultas. La primera capa no se contabiliza porque es la capa de entrada, la particularidad de estas redes es que las conexiones se realizan entre capas consecutivas, es decir, no hay conexiones entre capas no consecutivas, como lo muestra la Figura 1.

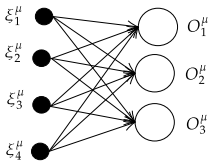

FIG. 1: Red feed forward

Puede computarse la salida de la red mediante la siguiente ecuación:

$$O_i^\mu = g(h_i) - g\left(\sum_k w_{ik}\xi_k\right) \tag{1}$$

En la ecuación 1, la relación o conexiones entre las neuronas de entrada (ξ_k) y las neuronas de salida (O_i^μ) se representa por medio de los ponderadores w_{ik}, donde i es el índice de la neurona de salida y k es el índice de la neurona de entrada. El cáculo de la salida dependera de la función de activación g y de la función de propagación h_i.

Al utilizar una red feed forward autoencoder, se busca que la red aprenda a reconstruir la entrada en la salida, es decir, que la salida sea igual a la entrada. Por lo tanto, la red debe aprender a comprimir la información de entrada en una representación más pequeña, para luego descomprimir la información y obtener la salida. Como trabajaremos con una solo capa oculta, la red tendrá la arquitectura de la figura 2, donde V_j es la función de activación de la capa oculta.

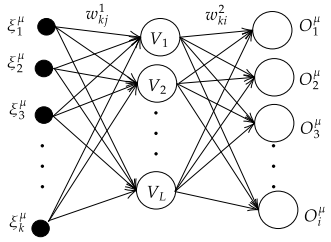

FIG. 2: Red feed forward autoencoder

Este tipo de redes, tienen la particularidad que la capa de entrada y la capa de salida deben tener la misma cantidad de neuronas, mientras que las capas intermedias deben tener una cantidad menor de neuronas. Esto obliga a la red a encontrar patrones entre los elementos de entrada y salida, de forma que aprenda a presentar la información comprimida y permitiendo asi reducir la dimensionalidad de los datos.

II. ARQUITECTURA DE RED NEURONAL

En este trabajo, se implementará una red feed forward autoencoder al corpus Fashion-MNIST, el cual consiste en un conjunto de imágenes de ropa en escala de grises de 28x28 pixeles, conformadas por:

```
0: 'T-shirt/top',
1: 'Trouser',
2: 'Pullover',
3: 'Dress',
4: 'Coat',
5: 'Sandal',
6: 'Shirt',
7: 'Sneaker',
8: 'Bag',
9: 'Ankle boot'
```

Donde el principal objetivo es clasificar una imagen de entrada en una de las 10 categorías. En la figura 3 se muestra la arquitectura de la red. Donde se observa que a la imagen inicial en la capa 1 se la aplica un *flatten* para transformar la imagen de $28x28$ en un vector de 784 elementos, luego pasa por la primera capa oculta de $n1$ neuronas la cual al inicio se le aplica un modulo *lineal*, luego se le aplica una función de activación *ReLU* y finalmente se le aplica un modulo *dropout* con una probabilidad de 0.2.

Luego, la salida de la primera capa oculta pasa por la segunda capa oculta de $n2$ neuronas que tiene el mismo comportamiento que la primera capa oculta. Finalmente, el flujo de la red llega a la capa final que tiene 10 neuronas cada una representando una de las categorías de Fashion-MNIST. Asi el output final es un vector de 10, en este ejemplo particular se puede ver que como input se paso una imagen de Ankle boot y como el output el valor más alto se da en la posición 9 significa que la red clasifico correctamente la imagen.

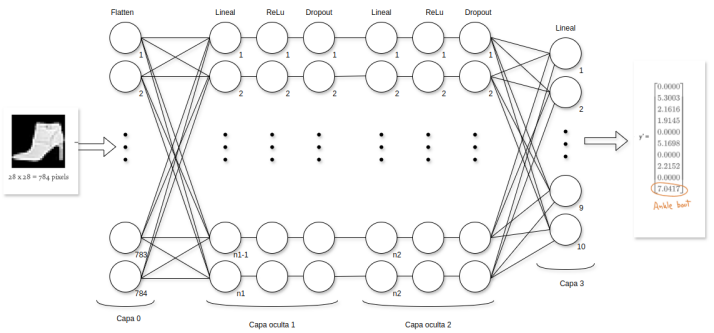

FIG. 3: Arquitectura de la red neuronal

Entonces el objetivo del trabajo sera entrenar esta red usando PyTorch como función de pardida se utiliza el error

cuadrático medio y el método de descenso por el gradiente estocástico como algoritmo de optimización (minimización), con un learning rate de $1e-3$ que indica 'de que tamaño' dar el paso para que el algoritmo converga a la solución. Lo idear es hacerlo de forma proporcional al gradiente, es decir, si el gradiente es grande el paso será grande y si el gradiente es pequeño el paso será pequeño.

El proceso de descenso por el gradiente estocástico se realiza en mini-batches, es decir, se toma un subconjunto de los datos de entrenamiento y se calcula el gradiente para ese subconjunto, luego se actualizan los pesos de la red y se toma otro subconjunto de datos y se repite el proceso. Esto se hace para evitar que la red se sobreajuste a los datos de entrenamiento y no pueda generalizar a datos nuevos.

Como base se utilizaran los siguientes hiperparámetros: $n1 = 128$, $n2 = 64$, 30 épocas y un batch size de 100 con un optimizador Stoachastic Gradient Descent (SGD) con un learning rate de $1e-3$. Luego se realizaran pruebas variando los hiperparámetros para ver como afectan al desempeño de la red, en particular se cambiara el optimizador por Adam y por ultimo se cambiara las epocas a 60, en la siguiente sección presentaremos estos resultados

III. RESULTADOS

Aqui presentaremos los resultados obtenidos al entrenar la red neuronal en los diferentes modelos planteados. Los hiperparámetros son:

- Modelo 1: $n1 = 128$, $n2 = 64$, 30 épocas y un batch size de 100 con un optimizador Stoachastic Gradient Descent (SGD) con un learning rate de $1e-3$.
- Modelo 2: $n1 = 128$, $n2 = 64$, 30 épocas y un batch size de 100 con un optimizador Adam con un learning rate de $1e-3$.
- Modelo 3: $n1 = 128$, $n2 = 64$, 60 épocas y un batch size de 100 con un optimizador Adam con un learning rate de $1e-3$.

En la tabla I se presentan los resultados obtenidos para cada modelo. Donde se puede observar que el modelo 2 y 3 tienen un mejor desempeño que el modelo 1, esto se debe a que el optimizador Adam es más eficiente que el SGD, ya que este último no tiene en cuenta la dirección del gradiente, sino que se mueve en la dirección opuesta al gradiente. Por otro lado, se observa que el el modelo 2 y el 3 tienen un desempeño similar, lo que significa que aumenta las epocas de entrenamiento no necesariamente mejora el desempeño de la red.

Entonces, se puede concluir que el modelo 2 es el mejor modelo, ya que tiene un mejor desempeño y menor tiempo de entrenamiento que el modelo 3.

Modelos	Loss	Acuracy
Modelo 1	0.51%	81.0%
Modelo 2	0.32%	89.3%
Modelo 3	0.37%	89.1%

TABLE I: Resultados del entrenamoento de la red neuronal

A. Modelo Base

B. Modelo con optimizador Adam

C. Modelo con 60 épocas

IV. CONCLUSIONES

* kevin.mansilla@mi.unc.edu.ar