Red Neuronal Autoencoder

Ramiro Santamaria* and Lijandy Jiménez Armas**
Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires and Instituto de Matemática Aplicada del Litoral
(Dated: 5 de febrero de 2024)

En el siguiente trabajo se presenta la implementación de redes neuronales autoencoder de una sola capa oculta con N neuronas para la reproducción de imágenes del set de datos de Fashion-MNIST. En particular se utilizaron $N = 64, 128, 256, 512$, observandose una mejor reproducción de las muestras de entrada a medida que se aumenta N.

I. INTRODUCCIÓN

Los autoencoders, en el ámbito del aprendizaje profundo, son redes neuronales diseñadas para incorporar la esencia de datos complejos a través de un proceso de codificación y decodificación. Este enfoque se basa en la capacidad de la red para aprender representaciones eficientes y compactas de los datos de entrada.

La estructura básica de un autoencoder comprende un codificador y un decodificador. El codificador transforma la entrada original en una representación de dimensionalidad reducida, llamada "capa oculta". Esta capa busca capturar las características más importantes de los datos. Posteriormente, el decodificador intenta reconstruir la entrada original a partir de esta representación más compacta.

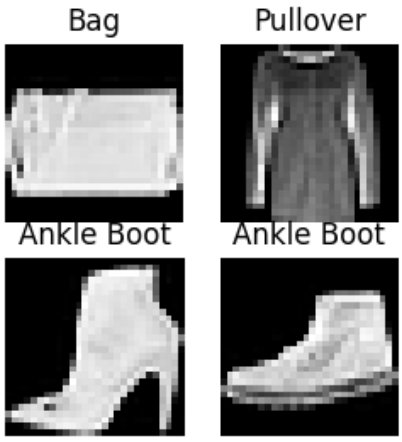


Figura 1: Ejemplos del tipo de muestras del dataset Fashion-MNIST

Aunque la función de un autoencoder es replicar los datos de entrada, su característica más importante radica en la capacidad para descubrir patrones intrínsecos y estructuras subyacentes durante el proceso de entrenamiento. Esto le permite a los autoencoders ser utilizados en diversas aplicaciones, desde la simplificación de datos hasta la creación de representaciones más eficientes.

En éste trabajo se estudiaran diferentes tipos de autoencoders con sólo una capa oculta, variando la dimensionalidad de esta. Para esto entrenaremos las redes con el set de datos Fashion-MNIST, que consta con 60000 datos de entrenamiento y 10000 datos de testeo. Cada unos de ellos es una imagen de $28x28$ píxeles en escala de grises y que pueden pertenecer a 10 clases diferentes 1.

II. METODOLOGÍA

Como se menciono en la sección anterior, en este trabajo se entrenó diferentes modelos de autoencoder con una capa oculta con el fin de replicar el set de datos de Fashion-MNIST. Para esto se comparó el rendimiento de redes que tuviese $N = 64, 128, 256, 512$ neuronas en su capa oculta, que es la encargada de redimensionalizar los datos de entrada que tienen 784 elementos ($28x28$ píxeles). El modelo de autoencoder utilizado en este trabajo se presenta en el esquema de la Fig.2.

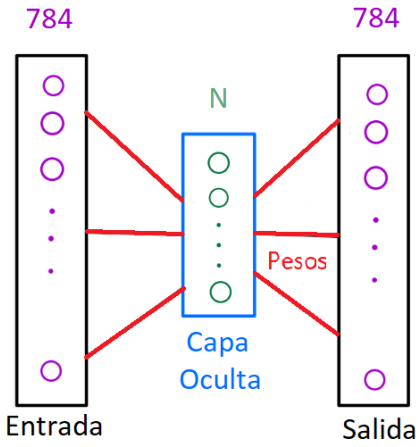


Figura 2: Esquema del modelo de autoencoder utilizado

El modelo contará con una capa de entrada de un vector de longitud 784, que son los píxeles totales en cada una de las imágenes, a la cual se le aplica una transformación lineal que lleva al vector a tener una longitud N. Para que nuestra red sea capaz de aprender patrones y relaciones complejas es necesario aplicar algún tipo de no linealidad en nuestra red. Con éste fin luego de la capa con N neuronas aplicamos una función Unidad Lineal Rectificada ($ReLU(x)$):

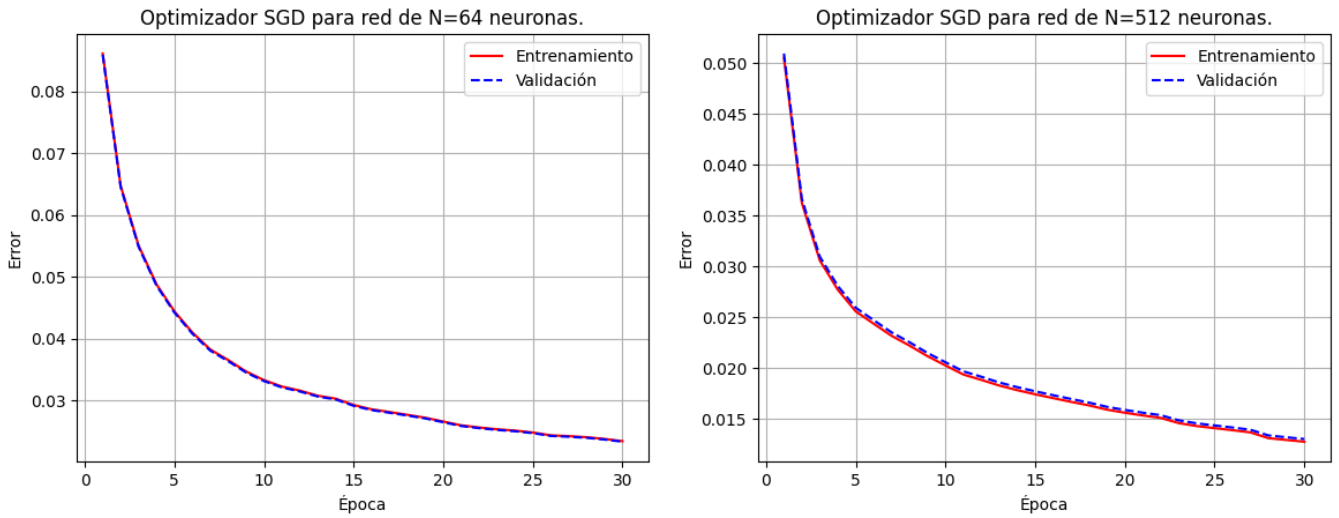


Figura 3: Evoluciones de los errores de entrenamiento y validación a lo largo de las épocas para $N = 64$ neuronas en la capa oculta (panel izquierdo) y $N = 512$ (panel derecho).

$$ReLU(x) = max(0, x) \tag{1}$$

es decir que si el valor de la neurona es > 0 se mantendrá el mismo valor, pero si es < 0 la neurona pasará a valer cero.

Una técnica se se suele utilizar para evitar el sobreajuste (o overfitting) es la función Dropout que apaga ciertas neuronas aleatoriamente con una probabilidad p (0,1 en este trabajo), de esta manera se ayuda a prevenir la dependencia entre unidades neuronales. Con esta idea, antes de volver a aplicar una transformación lineal que nos lleve de un vector de longitud N a uno con longitud 784, aplicamos la función Dropout. Como resultado obtenemos en la salida un vector de longitud 784 al que por último se le aplica nuevamente la función $ReLU(x)$. El código de la red autoencoder utilizada en este trabajo se muestra en el Apénd.A.

A los elementos de salida se los compara con los valores esperables, que para un autoencoder son los datos de entrada, y se calcula que tan "lejos" están uno del otro. En particular en este trabajo, para comparar el error de la predicción de la imagen formada por el modelo y la imagen original, se utilizó el error cuadrático medio (ECM):

$$ECM = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2 \tag{2}$$

con N el número de muestras, $\hat{Y}_i$ el valor obtenido por el modelo e Y_i el valor esperado, que para el autoencoder es igual al dato de entrada (la imagen).

Luego es necesario un optimizador, que es el que se encarga de ajustar los pesos de las transformaciones que

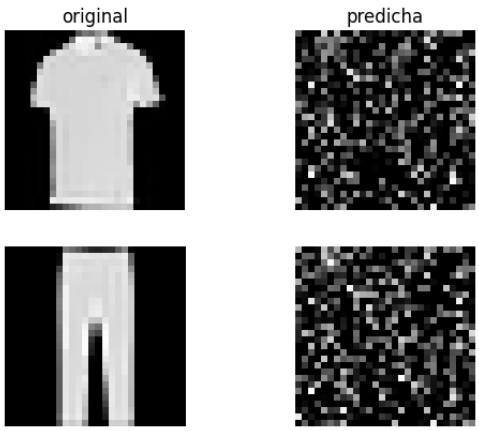


Figura 4: Ejemplo de la acción de una red sin entrenar sobre un par de muestras.

vinculan a las neuronas de las diferentes capas sucesivas entre sí. Estos ajustes pueden ser representados por la ecuación:

$$W_{ik}^{Nuevo,P\leftrightarrow P+1} = W_{ik}^{Viejo,P\leftrightarrow P+1} + \Delta W_{ik}^{P\leftrightarrow P+1} \tag{3}$$

con el miembro de la izquierda representando los pesos nuevos que vinculan a la neurona i de la capa $P+1$ con la neurona k de la capa P. Los pesos se ven modificados incrementando los valores viejos de los W_{ik} en un factor ΔW_{ik}.

Los valores de ΔW_{ik} dependerán del tipo de optimizador que se utilice para entrenar el modelo. En particular en éste trabajo se empleó el modelo de descenso por el gradiente estocástico (SGD por sus siglas en inglés). La idea es que las modificaciones de los parámetros atarán

dadas por:

$$\Delta W_{ik} = -\eta \frac{\partial E}{\partial W_{ik}} \tag{4}$$

con η el hiperparámetro que caracteriza el paso del tamaño de aprendizaje, en particular se usó 0,5, y las derivadas parciales se realizan sobre la función error utilizada (ECM) con respecto a su dependencia con los parámetros W_{ik}. Lo que nos permite buscar un mínimo local de una función de muchas variables.

Inicialmente los parámetros de los pesos W_{ik} se les asigna un valor aleatorio para luego ir siendo modificados durante el proceso de aprendizaje. En la Fig.4 se muestra la acción de la red sin entrenar sobre, en donde vemos que no hay ningún patrón aparente sino más bien parece ruido blanco.

A los datos de entrenamiento del set de Fashion-MNIST se los dividió en 50000 que van a ser usados efectivamente para entrenamiento, y se dejó 10000 datos para la validación del modelo. Es decir que a medida que la red va siendo entrenada se comparan los errores del autoencoder obtenidos con datos con los que fue entrenado (50000 de entrenamiento) y datos que no utilizó para aprender (10000 de validación). Cada vez que el modelo termina de analizar todos los datos de entrenamiento se lo denomina un época de aprendizaje. Al final de cada época se calculan y se guardan el error con los datos de entrenamiento y validación para ser graficados posteriormente. Una vez que el entrenamiento de la red finaliza se evalúan los datos de testeo (10000) sobre los modelos y se calcula su error.

III. RESULTADOS Y DISCUSIONES

En ésta sección compararemos los resultados obtenidos para los diferentes modelos, es decir variando la cantidad de neuronas presentes en la capa oculta.

Inicializamos el proceso de aprendizaje para los distintos modelos con diferentes valores de neuronas en su capa oculta (64, 128, 256 y 512) separando los datos de entrenamiento en la cantidad de lotes en los que va aprendiendo la red. En el procesamientos de cada lote la red actualiza los pesos de sinapsis entre las neuronas de cada capa, y este proceso se realiza hasta que se analizan todos los datos de entrenamiento (50000) dando por finalizada una época.

En éste trabajo se utilizaron lotes con un tamaño de 1000 muestras y se utilizaron 30 épocas de entrenamiento para todos los modelos. Se utilizó esta cantidad de épocas ya que el error de validación entre época y época no resultaba significativo para valores superiores. Es decir que se apreciaba cierto estancamiento en el error.

Tal comportamiento lo podemos apreciar en los gráficos de la Fig.3, en donde se muestran la evolución a lo largo de las épocas de los errores de entrenamiento (línea continua roja) y validación (línea punteada azul) utilizando el método SGD para los autoencoder con $N = 64$ y $N = 512$ en el panel izquierdo y derecho respectivamente.

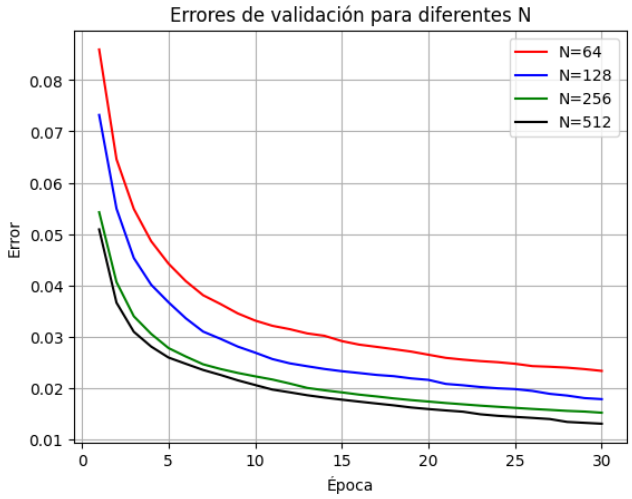


Figura 5: Evolución de los errores de validación para los diferentes modelos estudiados de cantidad de neuronas en la capa oculta.

Podemos hacer un par de observaciones respecto de la Fig.3. En primer lugar vemos que la mayor mejora porcentual de ambos modelos se realiza en las primera épocas de entrenamiento.

El modelo de $N = 64$ muestra una mejora de la época 1 a 2 de 0,0879 a 0,0686 respectivamente, que representa una mejora porcentual de 22 %. Mientras que de la época 29 a 30 sólo hay una mejora de 0,025 a 0,0247, que representa una mejora de sólo 1,2 %.

Para el modelo de $N = 512$ vemos una mejora desde la época 1 a 2 de 0,0501 a 0,0357 respectivamente, que representa una mejora del 28,7 %. Y de la época 29 a 30 sólo hay una mejora de 0,0131 a 0,0129, que porcentualmente es una mejora de 1,5 %.

Por el bajo valor porcentual de mejora entre estas últimas se decidió utilizar 30 épocas para entrenar a los distintos modelos.

Por otro lado vemos que la evolución del error de validación, que evalúa datos que la red no utilizó para entrenar, nunca se separa demasiado de los errores de entrenamiento. Este comportamiento es también exhibido por los otros modelos con $N = 128$ y $N = 256$ neuronas en sus capas ocultas.

La comparación entre los errores de validación para los diferentes modelos estudiados se pueden ver en la Fig.5. Podemos notar que a medida que vamos aumentando el número de neuronas en la capa oculta el error de validación obtenido en la primer época va disminuyendo. Es decir el modelo con $N = 64$ (curva roja) tiene un mayor

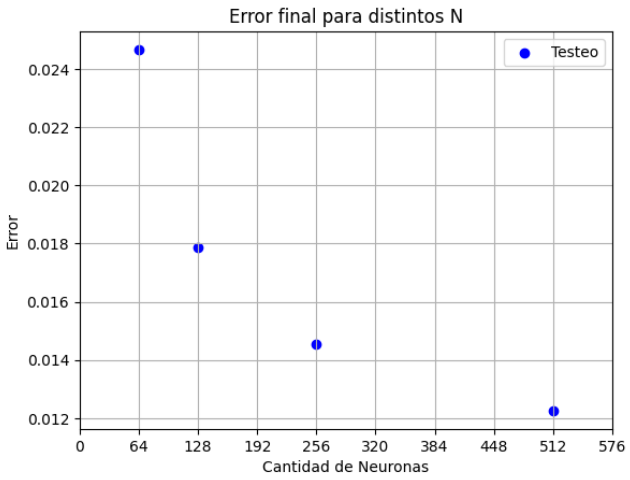


Figura 6: Errores de los datos de testeo para los diferentes modelos entrenados.

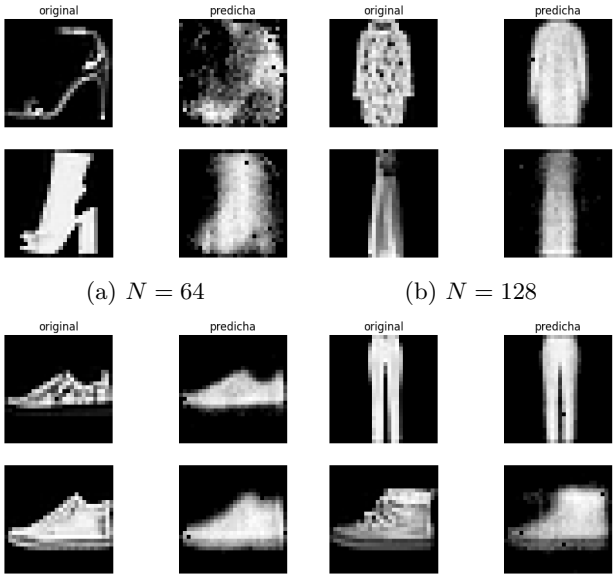
error de validación en la primer época que el modelo para $N = 128$ (curva azul), a su vez, éste presenta mayor error que el modelo $N = 256$ (curva verde), y por último éste presenta mayor error que el modelo $N = 512$ (curva negra). Comportamiento también observado en la época 30, al finalizar el proceso de aprendizaje.

Finalmente en la Fig.6 se muestran los errores obtenidos para los modelos finales, es decir los modelos que ya fueron entrenados durante las 30 épocas. Para esto se tomaron los datos de testeo, que no fueron utilizados ni para entrenamiento ni tampoco para validación, y se los utilizó para generar las predicciones del modelo. Podemos notar la disminución del error a medida que vamos aumentando la cantidad de neuronas en la capa oculta. Comportamiento que podíamos intuir por los errores de la última época de la Fig.5. Los valores de los errores de testeo para los modelos entrenados se muestran en la Tab.I

N	64	128	256	512
Error	0,0241	0,0179	0,0146	0,0123

Tabla I: Errores de los modelos entrenados para los datos de testeo.

Por último para tener una imagen cualitativa del modelo en la Fig.7 se muestran los resultados obtenidos evaluando los modelos entrenados con datos del conjunto de testeo. Vemos que en el panel Fig.7a, que muestra el modelo con $N = 64$, no presenta una definición predictiva lo suficientemente buena como para poder catalogar la figura, pero hay cierta correspondencia entre los píxeles activados por el modelos en comparación con la imagen original. Duplicando la cantidad de neuronas en la capa oculta, Fig.7b, ya es posible distinguir a que categoría



(a) $N = 64$ (b) $N = 128$

(c) $N = 256$ (d) $N = 512$

Figura 7: Comparación entre el funcionamiento del autoencoder con diferentes cantidades de neuronas N en la capa oculta.

pertenecen las imágenes predichas. Vemos que el modelo define bien los contornos de las figuras pero los detalles de la misma no son tenidas en cuenta.

Avanzando con la cantidad de neuronas de la capa oculta, $N = 256$ (Fig.7c) y $N = 512$ (Fig.7d), vemos que los contornos de las figuras están perfectamente definidos. Incluso en la región que hay entre las entre las "mangas" del pantalón (o perneras). También vemos presentes algunos píxeles que permanecen oscuros.

IV. CONCLUSIONES

En éste trabajo se estudiaron modelos de autoencoder con una sola capa oculta, variando la cantidad de neuronas presentes $N = 64, 128, 256, 512$. A los modelos se los entrenó con el set de datos de Fashion-MINST que consta de 60000 datos de entrenamiento y se utilizaron los 10000 datos de testeo para calcular el error final de los distintos modelos.

Para cuantificar las diferencias entre las imágenes esperadas y las predicciones que realizó el modelo se utilizó el error cuadrático medio (ECM). Como método de optimización de los parámetros se optó por el método de descenso por en gradiente con una tasa de tamaño de aprendizaje de 0,5.

Se observó que a lo largo de las épocas de entrenamiento los modelos realizan la mayor tasa de aprendizaje durante las primeras épocas, y luego la tasa de corrección del error tiende a aplanarse.

Por último se evaluó los modelos ya entrenados con los datos de testeo, observándose una mejora en la representación de los datos de entrada para los modelos que presentan mayor cantidad de neuronas en la capa oculta.

* rami.santamaria92@gmail.com

** lijandy92@gmail.com

Apéndice A: Modelo

```
# Definimos el autoencoder con parámetros de entrada la cantidad
# de neuronas de la capa oculta y el porcentaje de dropout

class Autoencoder(nn.Module):
    def __init__(self,n,p):
        super(Autoencoder, self).__init__()
        self.flatten = nn.Flatten()

        # Procedimiento que aplica la red
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28,n),
            nn.ReLU(),
            nn.Dropout(p),
            nn.Linear(n,28*28),
            nn.ReLU(),

        )

    # Se aplica el procediento
    def forward(self, x):
        x = self.flatten(x)
        return self.linear_relu_stack(x)
```