Redes Neuronales: Práctico 4

Mauricio Mazuecos

1 Introducción

En el práctico se debe implementar un *Neural Autoencoder* para aprender el conjunto de datos MNIST de dígitos manuscritos (figura 1). Este conjunto son imágenes de $28px^2$ en blanco y negro con píxeles cuyo valor va entre 0 y 255.

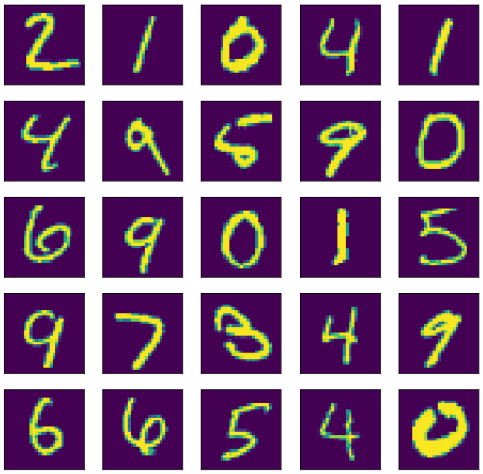

Figure 1: Ejemplos de datos del conjunto MNIST de dígitos manuscritos.

Un *Neural Autoencoder* es un modelo para reducción de dimensionalidad y específico a un conjunto de datos. La idea de los mismos es encontrar una representación densa y de baja dimensión que sea suficientemente expresiva para reconstruir la entrada del mismo. Luego, estas representaciones pueden ser usadas por otros modelos en lugar de representaciones como *one-hot* (en PLN) o para introducir features visuales en alguna tarea (como en el área de Diálogo Visual).

2 Implementación

La arquitectura está detallada en el figura 2. Para operar con el conjunto de datos, se transformaron las entradas de imágenes de $28px^2$ a vectores de 784 dimensiones. Luego estos son pasados por una capa densa (*Fully-Connected*) con función de activación ReLU para generar una representación interna de 64 dimensiones. Finalmente el flujo sigue a una capa de salida con 784 neuronas para reconstruir la entrada con función de activación ReLU.

El entrenamiento del modelo se basa en reconstruir la entrada a partir de una representación de menor dimensionalidad. Sean θ los pesos de la red, queremos encontrar unos pesos tales que minimicen una función de pérdida definida como:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=0}^{N}(X^{(i)} - h_\theta(X^{(i)}))^2 \tag{1}$$

donde N es el tamaño del conjunto de datos, $X^{(i)}$ es el i-ésimo ejemplo de entrenamiento y h_θ es una función (en este caso, una red neuronal) con parámetros θ.

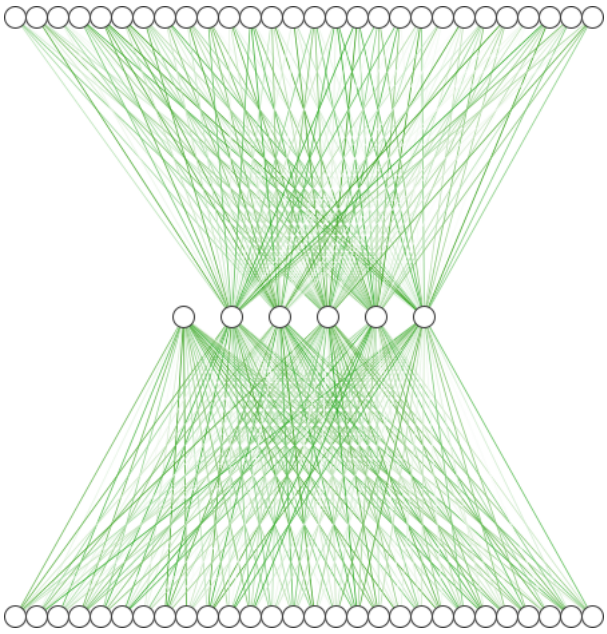

Figure 2: Arquitectura del Neural Autoencoder. Los datos flujen desde la entrada (arriba), se reducen de dimensionalidad y se reconstruyen en la última capa (abajo).

El modelo es entrenado durante 20 épocas empleando Adam como optimizador con una taza de aprendizaje $\alpha = 0.001$. En la figura 3 se muestra una visualización del grafo del cálculo del gradiente que se generó para esta arquitectura. El código estará disponible en `https://github.com/maurygreen/RedesNeuronales4`.

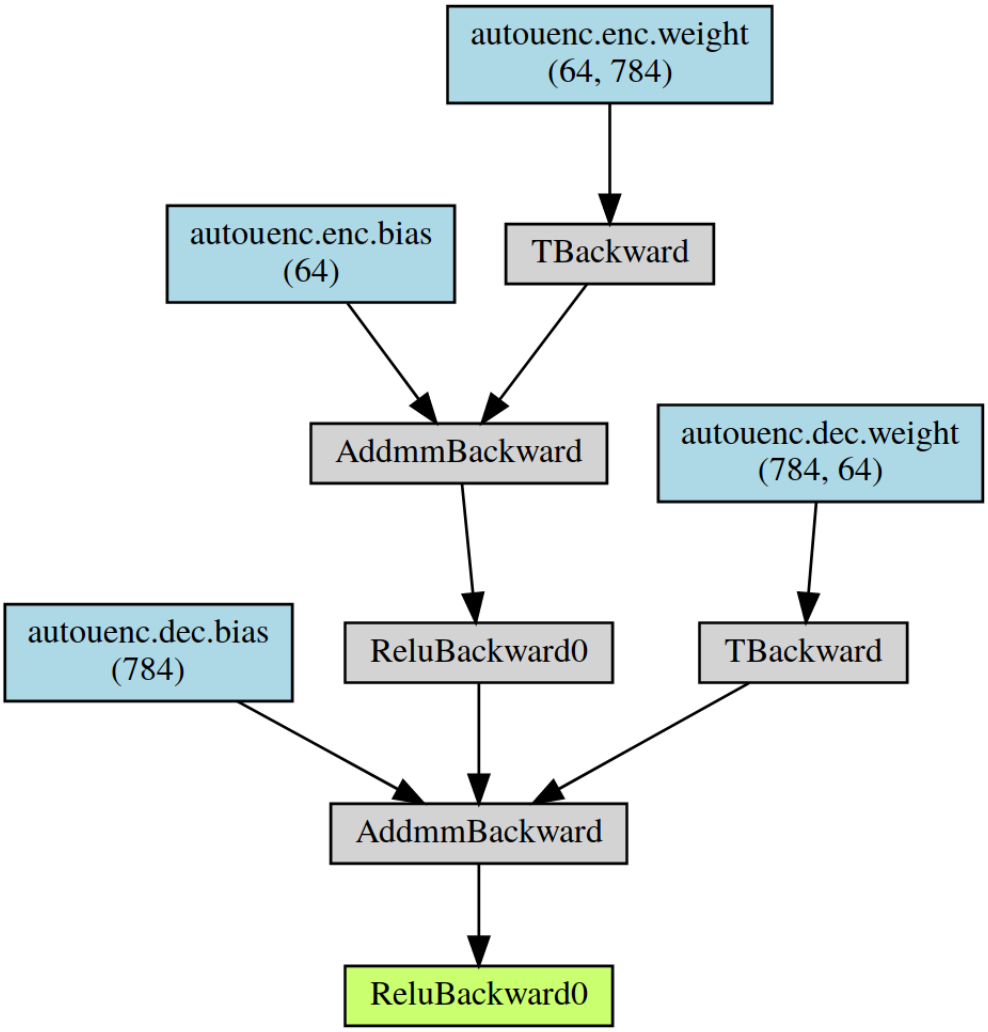

Figure 3: Visualización del grafo de operaciones para el cálculo del gradiente generada con TorchViz.

3 Resultados

El entrenamiento se llevó a cabo empleando un 90% del conjunto de datos como conjunto de entrenamiento y el 10% restante como conjunto de validación. En la figura 4 se puede ver que la función de pérdida a lo largo de las épocas decrece en ambas curvas, motivo por el cual se puede descartar un overfitting.

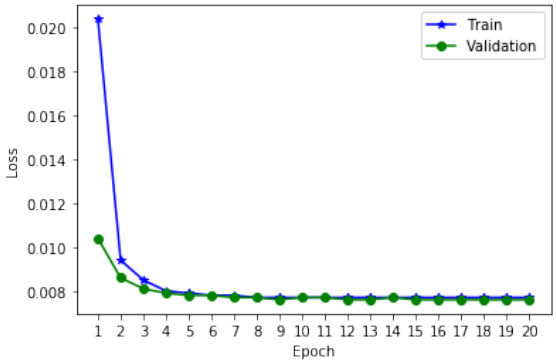

Figure 4: Función de pérdida en el conjunto de entrenamiento (azul) y validación (verde) a lo largo de las épocas de entrenamiento. Se puede observar que ambas curvas decrecen, motivo por el cual se puede afirmar que el entrenamiento no está cayendo en un caso de *overfitting*.

Finalmente en la tabla 1 se reporta la pérdida en el conjunto de test comparado a las otras particiones. En la figura 5 se pueden ver ejemplos de reconstrucciones de la entrada generadas por el *Autoencoder*.

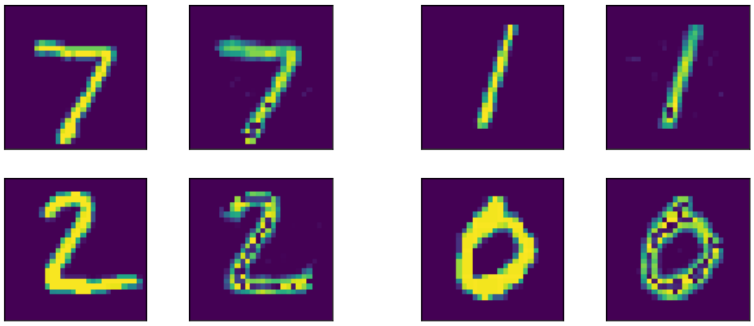

Figure 5: Ejemplos de reconstrucciones generadas por el *Autoencoder*.

	Train	Dev	Test
MSE	0.0077	0.0076	0.0075

Table 1: Valor de la función de pérdida en cada una de las particiones del conjunto de datos MNIST.

4 Conclusiones

Los *Neural Autoencoders* son modelos de reducción de dimensionalidad muy útiles para distintas tareas. En particular, el problema del conjunto de datos MNIST de dígitos manuscritos es uno en el que estos modelos se destacan y pueden encontrar buenas representaciones sin mayores complicaciones. Conseguir representaciones de baja dimensionalidad como esta pueden ayudar en varias tareas donde incluir features visuales es fundamental al mismo tiempo de ahorrar espacio de almacenamiento y cómputo.