Reducción de ruido con Autoencoders

Mauricio Pinto Larrea
(dept. Computer Science)
UTEC
Lima, Peru
mauricio.pinto@utec.edu.pe

Francesco Ucelli Meneses
(dept. Computer Science)
UTEC
Lima, Peru
francesco.uccelli@utec.edu.pe

Juan Manuel Navarro Nieto
(dept. Computer Science)
UTEC
Lima, Peru
juan.navarro@utec.edu.pe

I. Introducción

Los *Autoencoders* son utilizados en amplias áreas de *machine learning* (ML) como la reconstrucción de imágenes, mitigación de ruido y la creación de audio. Estos modelos buscan reducir datos característicos, de manera que puedan generar o reconstruir información en base a una entrada. En este trabajo se generó un modelo basado en autoencoders que logra eliminar el ruido de las imágenes. Se utilizó una base de datos que cuenta con 214 imágenes con fragmentos de texto, algunas con ruido y otras sin ruido. Los objetivos de este proyecto fueron los siguientes:

1) Implementar modelos de autoencoders basados en *Convolutional Neural Networks* (CNN) y *Multilayer Perceptron* (MLP).
2) Lograr mitigar el ruido de las imágenes utilizando los modelos implementados
3) Experimentar y hallar una arquitectura de red óptima
4) Analizar resultados y discutir observaciones

A continuación se hará una breve explicación conceptual de los autoencoders y las arquitecturas utilizadas, se detallará cómo se realizaron los experimentos y se presentarán los resultados.

II. Explicación

A. Arquitectura de la Red

Un autoencoder es un tipo de red neuronal utilizado para aprender la representación comprimida de data cruda. Estos son compuestos por submodelos de encoder y decoders, los cuales se encargan de comprimir el input y recrear el input respectivamente, en base a la versión comprimida que brinda el encoder. Tras el entrenamiento, ya no hay mas uso para el decoder, el cual queda descartado, mientras el encoder se guarda para uso futuro. Para el caso de la arquitectura de el CNN Autoencoder se muestra a continuación:

Los encoders pueden ser utilizados como una técnica de preparación de data con el objetivo de extraer características de data cruda (raw data), la cual puede ser utilizada para entrenar luego a otro modelo de machine learning.

B. Autoencoders en la extracción de características

Un autoencoder, como mencionamos previamente, es una red neuronal que busca aprender una representación comprimida de algún input, en otras palabras, es entrenado para tratar de imitar el input como output [10]. Estos modelos son de aprendizaje no supervisado, a pesar de que se entrenen utilizando métodos supervisados, por lo que se le refiere como de auto-supervisión (self-supervised).

Los autoencoders son usualmente entrenados como parte de un modelo mas amplio con el afán de recrear el input, por ejemplo: $var = model.predict(X)$. El diseño de estos modelos adredemente lo hace mas trabajoso debido a que restringe la arquitectura a un bottleneck (cuello de botella) en el punto medio del modelo en donde se reconstruye el input de la data.

En este caso en particular, una vez que el modelo esta fiteado, la reconstrucción puede ser desechada y el modelo hasta el bottleneck puede ser usado, esto ya que el output en el punto del bottleneck es un vector de tamaño fijo que provee una representación comprimida del input.

Por lo general, están restringidos de manera que les permiten copiar solo aproximadamente y copiar solo la entrada que se asemeja a los datos de entrenamiento. Debido a que el modelo se ve obligado a priorizar qué aspectos de la entrada deben copiarse, a menudo aprende propiedades útiles de los datos.[10] En otras palabras, la data del input del dominio puede luego ser usada en el modelo, y el output del model en el bottleneck puede ser utilizado como un feature en un modelo de aprendizaje supervisado, tanto en visualización como en reducción de dimensiones.

III. Dataset

La data con la que se trabajó incluye imágenes de fragmentos de texto, con distintos tamaños y con la adición de ruidos distintos. Se contó con dos carpetas de entrenamiento, train y test, las cuales contaron con 214 imágenes de tipo jpg, cada una conteniendo un fragmento de texto. Cada nombre de los archivos representa un índice, de modo que el archivo $1.jpg$ es el mismo que el que podemos encontrar en la carpeta auxiliar $train_cleaned$, en el cual se encuentran las imágenes originales pero sin la presencia de ruido, las cuales serán utilizadas al final para corroborar la precisión de nuestro modelo.

IV. Experimentación

Los experimentos para el proyecto fueron realizados en un ambiente de *kaggle*, bastante similar a *Colab*. De esta manera se tuvo acceso a recursos de memoria y procesamiento (incluyendo GPU) en la nube, por lo que se pudo hacer

pruebas con dispositivos CUDA. Se implementaron los modelos mencionados anteriormente utilizando la librería *PyTorch* tanto para los modelos MLP como para los CNN. Las arquitecturas de cada uno se encuentran resumidas a continuación:

```
// CNN
CNN_Autoencoder(
    (conv1): Conv2d(1, 32, kernel_size
       =(4, 4), stride=(2, 2), padding
       =(1, 1))
    (conv2): Conv2d(32, 4, kernel_size
       =(3, 3), stride=(2, 2), padding
       =(1, 1))
    (fc): Linear(in_features=10,
       out_features=6272, bias=True)
    (t_conv1): ConvTranspose2d(4, 16,
       kernel_size=(2, 2), stride=(2, 2))
    (t_conv2): ConvTranspose2d(16, 1,
       kernel_size=(2, 2), stride=(2, 2))
)

// MLP
MLP_Autoencoder(
  (Encoder): MLP_Encoder(
    (mlp): Sequential(
      (0): Linear(in_features=48400,
         out_features=128, bias=True)
      (1): ReLU()
      (2): Linear(in_features=128,
         out_features=64, bias=True)
      (3): ReLU()
      (4): Linear(in_features=64,
         out_features=12, bias=True)
      (5): ReLU()
      (6): Linear(in_features=12,
         out_features=3, bias=True)
    )
  )
  (Decoder): MLP_Decoder(
    (mlp): Sequential(
      (0): Linear(in_features=3,
         out_features=12, bias=True)
      (1): ReLU()
      (2): Linear(in_features=12,
         out_features=64, bias=True)
      (3): ReLU()
      (4): Linear(in_features=64,
         out_features=128, bias=True)
      (5): ReLU()
      (6): Linear(in_features=128,
         out_features=48400, bias=True)
      (7): Sigmoid()
    )
  )
)
```

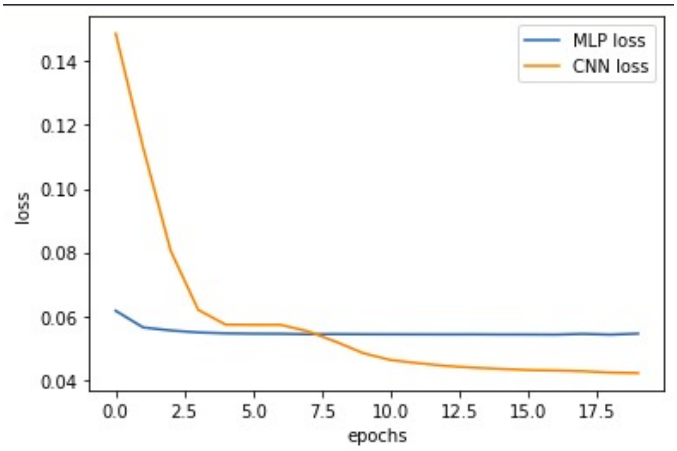

Fig. 1. Curva de pérdida VS número de épocas. Modelo basado en CNN (Izquierda) y modelo basado en MLP (derecha)

TABLE I
VALORES DE ERROR PARA CADA MODELO EN CADA EJECUCIÓN DEL K-FOLD CROSS VALIDATION CON DISTINTOS VALORES DE k

	k = 20	k = 40	k = 60	k = 80
Error MLP	0.0521	0.0523	0.0527	0.0534
Error CNN	0.0332	0.0583	0.0567	0.0668

Los errores calculados en la etapa de entrenamiento para cada modelo se encuentran representados en la Figura 1

Se pudo remover el ruido de las imágenes de prueba exitosamente, aunque esto implicó pérdidas de la información en las imágenes. Algunas imágenes generadas se pueden observar en la Figura 2.

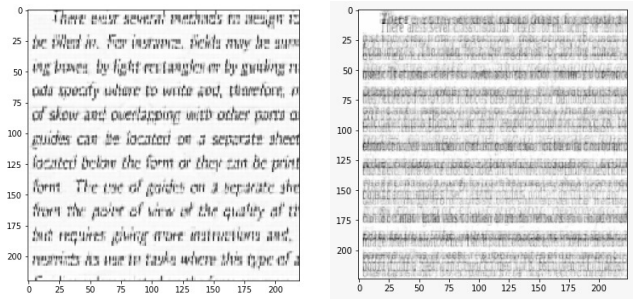

Fig. 2. Imágenes de prueba reconstruidas utilizando el modelo basado en CNN (Izquierda) y el modelo basado en MLP (Derecha)

Además, se experimentó modificando los valores de k al realizar el *k-fold cross validation*. Los resultados se observan en la Tabla I

Por último, los tiempos de ejecución medidos para cada valor de k se observan en la Tabla II

TABLE II
TIEMPOS DE EJECUCIÓN PROMEDIO PARA CADA MODELO EN 20 ÉPOCAS

	MLP model	CNN model
Tiempo (s)	39.1455	24.7626

V. CONCLUSIONES

Como se puede observar en la experimentación, a pesar de no tener una imagen nítida como en la original nuestro modelo logra reducir el ruido de manera considerable. Se puede observar la gran diferencia entre los resultados de un modelo de autoencoder con CNN contra uno con MLP, siendo el de CNN el que nos brinda resultados mas ajenos a lo deseado. Por otro lado, podemos observar que el modelo basado en MLP, a pesar de tener un error bastante bajo, a la hora de visualizar los resultados no logra lo esperado. Sin embargo, utilizando CNN a pesar de brindarnos con imágenes mas nítidas, el definir la arquitectura es mucho mas complejo.

VI. REFERENCIAS

1 Sathyanarayana, Shashi. (2014). A Gentle Introduction to Backpropagation. Numeric Insight, Inc Whitepaper.
2 https://scriptreference.com/neural-networks-from-scratch/neural-network-gradient-descent
3 http://alexlenail.me/NN-SVG/index.html
4 https://patrickhoo.wixsite.com/diveindatascience/single-post/2019/06/13/activation-functions-and-when-to-use-them
5 https://deepai.org/machine-learning-glossary-and-terms/softmax-layer
6 https://drive.google.com/file/d/1t47WOKOFnNuWd2hzvpu9h262L99f_Ab4/view
7 PyTorch vision models, https://pytorch.org/vision/stable/models.html
8 https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4
9 Kaggle Repository, https://www.kaggle.com/flrotm/proyecto6
10 Deep Learning (Adaptive Computation and Machine Learning series), page 502
11 https://machinelearningmastery.com/autoencoder-for-classification/