Estudio de una red neuronal clasificadora pre-entrenada por un autoencoder convolucional.

Redes Neuronales - Trabajo Final

Christopher Gabaldón - c_gabaldon@outlook.com

Estudiante de grado de Cs. Físicas en la Facultad de Ciencias Exactas y Naturales (FCEN) - UBA

Segundo Cuatrimestre 2023, Cátedra Francisco A. Tamarit

Facultad de Matemática, Astronomía, Física y Computación (FAMAF) - UNC

Resumen

En el presente trabajo se implementó una red neuronal clasificadora pre-entrenada con un autoencoder convolucional, empleando la base de datos *Fashion* del MNIST [1]. Se evaluaron posibles hiperparámetros a utilizar tanto en el autoencoder como en el clasificador de manera de minimizar el error del modelo y aumentar su capacidad de generalización. Adicionalmente se estudiaron dos fenómenos adyacentes: el uso de etapas *pooling* en las capas convolucionales y la actualización de los pesos sinápticos pre-entrenados.

1. Introducción

1.1. Motivación

Las redes neuronales convolucionales surgen para impulsar el desarrollo de la visión por computadora, permitiendo resolver estos problemas de manera mucho mas eficiente que empleando redes profundas totalmente conectadas. Hoy en día además encuentran aplicaciones en muchas áreas de la ciencia e industria. En este trabajo se estudiará su funcionamiento implementando un *autoencoder* convolucional para *preentrenar* un clasificador de imágenes, una aplicación en el campo de la visión por computadora.

1.2. Marco teórico

1.2.1. Redes Feed Forward como clasificador

Una red neuronal *feedforward (FF)*, es un tipo de arquitectura donde la información fluye en una única dirección, desde la capa de entrada hasta la capa de salida. Cada capa de la red cuenta con una cantidad N_l de neuronas($l = 1, .., L$) que procesan la información mediante la aplicación de funciones de activación. Esta estructura permite realizar la clasificación de elementos a través de la adaptación de los pesos de conexión entre las neuronas mediante algoritmos de entrenamiento.

En el marco del aprendizaje *supervisado*, se inyectan entradas ξ_κ que son vectores de dimensión (N_1+1) con $\kappa = 1, .., p$ elementos del conjunto de entrenamiento. Cada entrada tendrá una correspondiente salida etiquetada en dicho conjunto.

El algoritmo de aprendizaje que se empleará en este trabajo, es el descenso por el gradiente estocástico [2], donde se seleccionan al azar muestras en lotes (*mini-batches*) que se someten a la red y esta va corrigiendo sus pesos de manera que se minimice la función Error, siguiendo la ecuación:

$$\Delta\omega_{ij}^{nuevo} = \Delta\omega_{ij}^{viejo} - lr\frac{\partial E}{\partial \omega_{ij}} \tag{1}$$

Donde ω_{ij} representa la matriz de pesos "sinápticos", lr es la taza de aprendizaje y E es la función error.

Respecto a la función error, cuando la red funciona como clasificador es conveniente utilizar la función *Cross Entropy Loss (CEL)*.

Si la salida de la red es normalizada por la función Softmax, se puede interpretar a la respuesta de la red como una distribución de probabilidades. Así, si cada neurona de salida representa una clase, las salidas representarán la probabilidad con que la red clasifica cierta entrada. La función CEL para una clasificación de K clases excluyentes se define como:

$$CEL(P, Q) = -\sum_{i}^{K} P_i \log_2(Q_i) \tag{2}$$

Donde P es un valor binario que indica si la entrada pertenece o no a la *i-ésima* clase, y Q es la mencionada probabilidad predicha por la red. Se puede notar como la función penaliza proporcionalmente la discrepancia entre la predicción y la etiqueta real.

1.2.2. Redes Neuronales Convolucionales

En las redes neuronales convolucionales (RNC), la operación central es la correlación cruzada, operación similar a la convolución. La arquitectura de estas redes se caracteriza en aplicarle a la entrada dicha operación, contra tensores de dimensión menor llamados *núcleos* (kernels), el conjunto de núcleos conforma un

filtro. Los elementos de los núcleos van actualizando sus valores (equivalente a los pesos sinápticos de las redes FF) a lo largo de épocas de entrenamiento ajustándose para minimizar el error.

Una ventaja de este tipo de arquitectura es que preserva las dimensiones originales de la entrada pudiendo aprender patrones complejos.

Por otro lado, este tipo de redes son notablemente más eficientes que las redes completamente conectadas (RCC) en términos computacionales ([3] Cap 9). Como los núcleos se van 'deslizando' por la entrada, la cantidad de conexiones baja considerablemente respecto a las RCC, reduciendo la carga computacional. Además, las RNC no tienen *pesos fijos* como las RCC, sino que cada núcleo comparte su valor con todos los elementos de la entrada con los que interactúa. Por último, debido a la operación de correlación cruzada la red es *equivariante* frente a translaciones de la entrada, lo cual le da robustez y capacidad de generalización.

Otra operación empleada en una RNC es el *pooling*, ideado para reducir la dimensionalidad de la entrada al mismo tiempo que preserva sus características más relevantes, optimizando el cómputo. Además esto induce una *invariancia* frente a pequeñas translaciones de la entrada, que es beneficiosa para la generalización del modelo. En este trabajo se empleó el *max pooling* donde los elementos de una matriz de cierto tamaño se remplazan por el elemento con mayor valor de dicha matriz.

1.2.3. Autoencoder como preentrenamiento de un clasificador

Una red neuronal *autoencoder* es aquella que busca aprender la función identidad. En una primera etapa el *encoder* comprime la información de la entrada a capas de dimensión menor (espacio latente), luego en la etapa de *decoder* se reconstruye la entrada. Mediante transformaciones no lineales se fuerza a la red a aprender características distintivas de la entrada. Una métrica posible para determinar el desempeño de la red es el error cuadrático medio ECM.

Si se entrena un *autoencoder*, luego se pueden extraer los valores de sus pesos sinápticos para *preentrenar* otra red similar, y resolver el problema de la inicialización aleatoria de los pesos, volviendo más eficiente el proceso de aprendizaje [4].

Es interesante destacar que en un *autoencoder* con capas convolucionales (AC) las etapas de *pooling* pueden provocar una pérdida de información que perjudique la recreación de la entrada. De todas formas esto no significa que empeore la calidad del preentrenamiento si solo se emplean los pesos entrenados del *encoder* ([3] Cap. 9.3).

1.3. Objetivos de este trabajo

En este trabajo se buscará implementar un AC supervisado y se estudiará su comportamiento variando hiperparámetros involucrados, con la finalidad de seleccionar los que minimicen el ECM del modelo. Luego se usará la etapa de *encoder* preentrenada para implementar un clasificador, donde se le ingrese una imagen y este prediga a que categoría pertenece. Adicionalmente se evaluará el efecto del maxpooling y la actualización de los pesos sinápticos.

2. Implementación de los modelos

Tanto para el AC como para el clasificador se trabajó con *pytorch* empleando imágenes de ropa ByN de (28×28) px obtenidas de la base de datos $FashionMNIST$[1] con sus respectivas etiquetas (clases de ropa). Se construyó un conjunto de entrenamiento de 50.000 imágenes, además de conjuntos de validación y prueba con 10.000 imágenes cada uno.

2.1. Autoencoder convolucional

El esquema consistió en una fase de *encoder* provista por dos capas convolucionales con sus respectivas etapas de $MaxPooling$, donde se transformaron las dimensiones de la entrada (1,28,28) a n salidas de una capa lineal. En la Figura 1 se detallan las dimensiones de los núcleos y los pasos de la convolución, o *stride*, para cada uno de los núcleos empleados (16 en la primera capa, y 32 en la segunda). Las funciones de activación en estas capas fueron ReLU aplicando la técnica de *Dropout*, es decir desactivando ciertas conexiones con probabilidad p.

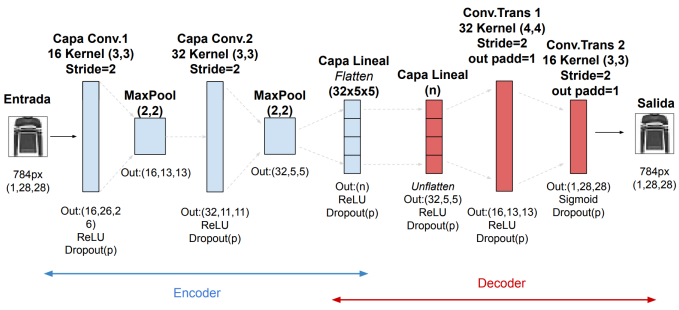


Figura 1: Esquema del AC. El encoder contó con dos capas convolucionales y una capa lineal, mientras que para el *decoder* se emplearon capas convolucionales traspuestas.

En la fase de *decoder* se transformó la dimensión n a dimensiones (1,28,28) empleando una capa lineal y dos capas convolucionales transpuestas. A todas las capas se le aplicó *dropout*, aplicándose en la segunda capa una función de activación *sigmoide*.

Para el entrenamiento, en cada época se inyectaron minilotes de tamaño $BS = 100$ del conjunto de

entrenamiento, donde la red iba modificando sus pesos usando el optimizador ADAM[5] que modifica la actualización de la Ec. (1) dotándola de cierta memoria. Luego, desactivando la maquinaria de optimización se calculó el error promedio de los lotes del conjunto de entrenamiento y de validación. Finalizado el entrenamiento, se sometió a la red a minilotes del conjunto de prueba donde se registró el error y se recreó la imagen de salida.

2.2. Clasificador pre-entrenado

Para implementar el clasificador se emplearon las capas de *encoder* del *autoencoder* (Sec. 2.1) entrenado. A su salida de dimensión n se conectó una capa lineal cuya salida tendría 10 dimensiones correspondiente a cada clase de ropa. La capa de salida fue provista de funciones de activación lineales (conveniente para la CEL), como enseña la Figura 2.

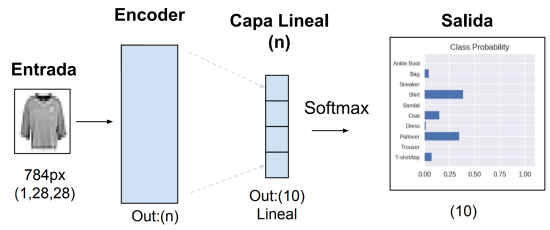


Figura 2: Esquema del clasificador. Se conectó el *encoder* entrenado a una capa lineal, y así clasificar la entrada.

El entrenamiento fue análogo al del *autoencoder* solo que en este caso se empleó la función CEL. A su salida la red entregaba un vector de probabilidades y se asoció al máximo elemento de dicho vector como la clase predicha. En cada minilote se contabilizó el número de predicciones correctas dividido el número total de las predicciones, esto es la *precisión* del modelo. Finalmente se tomó el promedio de todos los minilotes para graficar precisión y CEL *vs* época.

3. Resultados y discusiones

3.1. Autoencoder convolucional

Se estudió el comportamiento del autoencoder de la Sec. 2.1 al variar los hiperparámetros n,p y lr con el objetivo de seleccionar los que minimicen el ECM.

Fijando $p = 0.2, lr = 0.001$,variando n se entrenó al modelo en 30 épocas y luego se sometió a lotes del conjunto de prueba a la red. La Tabla 1, reporta los valores promedios del ECM (StD despreciables).

n	64	128	256	512
ECM	0.034	0.025	0.023	0.020

Tabla 1: Error del conjunto de prueba para diferentes n.

Se observó una disminución del error conforme aumentó n, aunque el tiempo de computo crecía. Esto se atribuyó a la complejización del modelo adquiriendo la facultad de aprender patrones más complejos de las imágenes.

Para analizar variaciones de p se fijaron lr=0.001 y n=128, por ser el que mejor balanceaba error-tiempo de cómputo. En la Tabla 2 se reportan los resultados del error para el conjunto de prueba en 30 épocas.

p	0.05	0.1	0.15	0.2
ECM	0.20	0.024	0.031	0.036

Tabla 2: Error del conjunto de prueba para diferentes p.

Así se determinó que $p = 0.05$ sería el valor óptimo dentro de los valores ensayados.

Finalmente para analizar variaciones del *learning rate* se fijaron p=0.05 y $n = 128$. La Tabla 3 detalla los resultados luego de 30 épocas.

lr	0.01	0.001	0.0001
ECM	0.042	0.020	0.023

Tabla 3: Error del conjunto de prueba para diferentes lr.

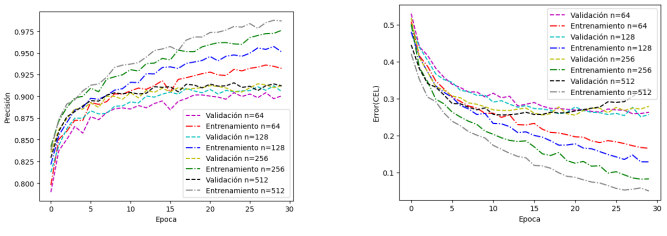
En el Apéndice se detalla un breve estudio del efecto de la etapa del *pooling* en el *encoder*, haciendo una comparación de dos modelos a iguales parámetros.

3.2. Clasificador pre-entrenado

Para el clasificador se usó el *encoder* preentrenado en 30 épocas con p=0.05, lr=0.001 y n=128. Actualizando con el optimizador ADAM tanto los pesos del *encoder* como los de la capa lineal (Ver Apéndice).

Se estudió el comportamiento del clasificador al variar los hiperparámetros n,p y lr con el objetivo de seleccionar los que minimicen la función CEL y mejoren la precisión del modelo como clasificador.

En la Figura 3 se observan las curvas resultantes de fijar p=0.05 y variar n para cada época.



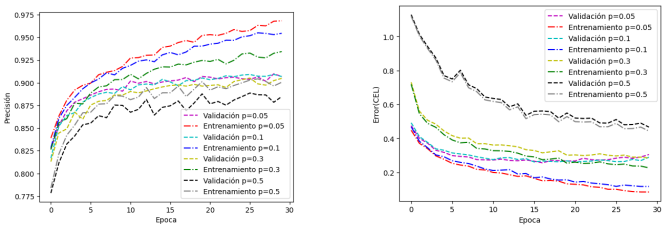
(a) Precisión *vs* época. **(b)** Función CEL *vs* época.

Figura 3: Curvas de error y precisión *vs* época del clasificador, para modelos con diferentes valores de n.

Se identificó una tendencia a minimizar el error y subir la precisión conforme aumenta n. De todas maneras se notó que al aumentar n existe una mayor brecha entre las curvas de validación y entrenamiento, habiendo sobreajuste a causa de complejizar el

modelo y aumentar la cantidad de pesos a aprender dificultando la generalización. Es importante mencionar que al pasar de dimensiones (28,28) a dimensión 10 era esperable notar el efecto de sobreajuste a diferencia del *autoencoder* donde las dimensiones entrada-salida eran iguales.

Por otro lado, se analizó el efecto de la variación de p fijando n=128 y lr=0.001. Se graficaron las curvas de error y precisión en función de las épocas de entrenamiento, representadas en la Figura 4.



(a) Precisión *vs* época. **(b)** Función CEL *vs* época

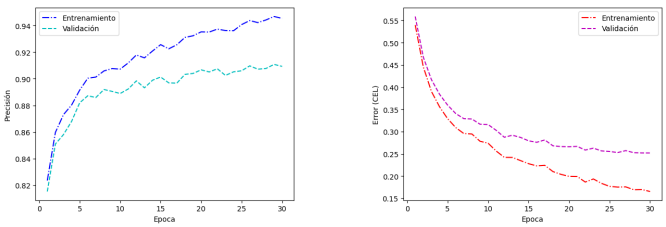
Figura 4: Curvas de error y precisión *vs* época del clasificador, para modelos con diferentes valores de p.

Al igual que en las variaciones de n, se observó la separación de las curvas de error aunque en este caso se redujo con los aumentos de p. En dichos casos el modelo empeora tanto en error como en precisión. Este fenómeno se atribuye nuevamente a la complejidad del modelo (al disminuir p se suprimen menos pesos por lo que se complejiza el modelo). Se determinó entonces como valor óptimo a $p = 0.1$ dentro de los ensayados.

Por último se estudió el efecto de la variación del lr fijando el resto de sus parámetros por sus valores óptimos. Se emplearon $lr = 0.01, 0.001, 0.0001$ y se notó un mejor desempeño tanto en la disminución de CEL como en la precisión del modelo para un número de 30 épocas trabajando con 0.001.

3.2.1. Evaluación del modelo

Con los parámetros establecidos como óptimos en la Sec. 3.2 se entrenó al clasificador durante 30 épocas. En la Figura 5 se observan las curvas de error y precisión en función de la época.



(a) Precisión *vs* época. **(b)** Función CEL *vs* época.

Figura 5: Métricas del modelo final.

La elección adecuada de n y p pudieron haber influido en la disminución de la diferencia entre las curvas de error y validación, asociada a la *varianza* del modelo. Se puede notar que con 15 épocas se obtendrían resultados similares, minimizando cómputos.

Para evaluar la capacidad de generalización del modelo se procedió a construir una *matriz de confusión* (Figura 6) en donde se aprecian las probabilidades de que la red prediga una dada clase. En este caso se trabajó con el conjunto de prueba, el cual era desconocido para la red ya entrenada.

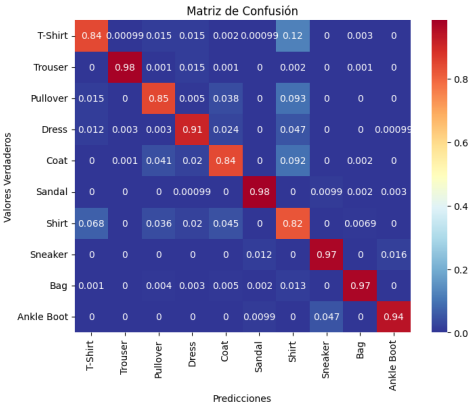


Figura 6: Matriz de confusión, donde la diagonal mayor representa la probabilidad de una correcta clasificación.

Se puede notar como la red clasifica correctamente con una probabilidad de 0.91 en promedio. Es importante destacar que tiene dificultades para clasificar prendas similares, como es el caso de las camisas (*shirt*) con los buzos (*pullover*) y remeras (*T-shirt*).

4. Conclusiones

Se cumplieron con los objetivos propuestos al lograrse implementar un clasificador preentrenado con un *autoencoder* convolucional. Además, a través de variaciones de los parámetros p, lr y n se estudió el comportamiento de la red.

Tanto en el *autoencoder* como en el clasificador se observó que un aumento de n, o una disminución de p, mejoraban el desempeño de la red a costo de un mayor tiempo de cómputo debido a la complejización del modelo.

Se empleó un clasificador con p=0.1, n=128, lr=0.001,BS=100 (parámetros óptimos) con el que se construyeron las curvas de error y precisión *vs* época. Adicionalmente se construyó una matriz de confusión donde se observó que en promedio la red predecía correctamente la clase con una probabilidad de 0.91, lo cual muestra una buena generalización del modelo. Se notó en dicha matriz la dificultad del modelo para distinguir prendas de vestir similares.

Además se estudió el efecto de las etapas de *pooling* y se observó que la calidad del *autoencoder* mejora notablemente al desactivarse estas etapas. Sin embargo, no se percibió este efecto en el clasificador.

Apéndice

Extracción del *pooling* del modelo

Motivado por la discusión del Cap 9.2 de *Deep Learning* [3] que infiere que las etapas de *pooling* pueden ser desventajosas en la construcción de un AC, se procedió a extraer del encoder a éstas, adecuando las dimensiones de entrada de la capa lineal y de la segunda capa convolucional (concatenando las capas convolucionales como las de la Figura 1 y usando un *flatten* para conectar la lineal).

Se procedió a entrenar un *autoencoder* con estas características empleando los mismos parámetros que el de la Sec. 2.1 (p=0.1, n=128, lr=0.001).

Luego de 30 épocas se calculó el ECM con el conjunto de prueba para ambos modelos, los resultados se reportan en la siguiente Tabla:

	Con *pooling*	Sin *pooling*
ECM	0.019	0.004

Además de la disminución del ECM, se notaba una mejor capacidad de recrear las imágenes de entrada como se aprecia en la Figura Ap.1 .

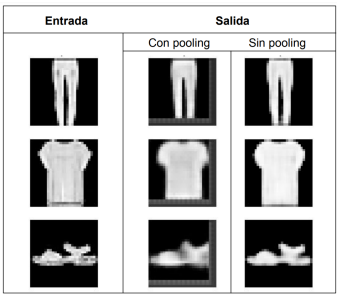
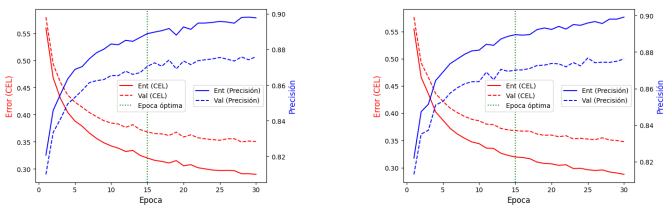


Figura Ap.1: Comparación de predicciones.

Luego se emplearon ambos modelos como preentrenamiento del clasificador, con el fin de observar si el efecto del *pooling* podría perjudicar a la tarea de clasificación.

Al igual que en la Sec.3.2 se graficaron las curvas de error y precisión en función de la época para ambos modelos, que se observan en la Figura Ap.2.



(a) Sin *pooling*. **(b)** Con *pooling*.

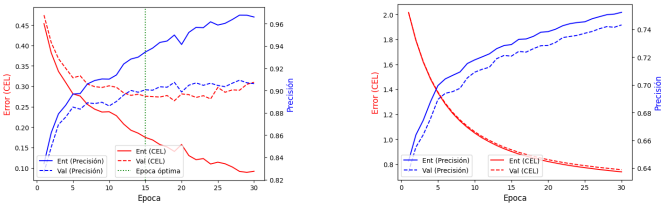
Figura Ap.2 :Función CEL y precisión *vs* época.

Se observa que las curvas para ambos modelos son similares. Esto manifiesta que sus desempeños como *autoencoders*, no se trasladan a sus desempeños como clasificadores.

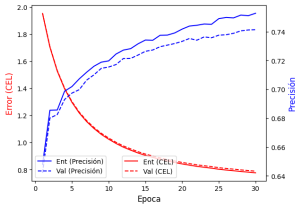
Pesos del *encoder* fijados

A la hora de usar los pesos del *encoder* en el clasificador, el método ADAM permitía actualizar todos los parámetros del clasificador, o actualizar únicamente los de la etapa lineal (es decir fijando los pesos del *encoder*). Adicionalmente se estudió la influencia de conectar el *encoder* al clasificador pero con pesos aleatorios (es decir sin entrenar).

Se graficaron las curvas error-precisión *vs* época para dichos modelos, como se ve en la Figura Ap.3.



(a) Sin fijar pesos. **(b)** Fijando pesos.



(c) Sin entrenar pesos.

Figura Ap.3 :Función CEL y precisión *vs* época.

Lo que se observa es que al fijar los pesos del *encoder* el modelo empeora en términos de error y precisión. Aunque se nota como desaparece el fenómeno de sobreajuste. En esta línea, también ocurre lo mismo sin entrenar los pesos del encoder, evidenciando la importancia del preentrenamiento.

Referencias

[1] Fashion MNIST, *Dataset*. `https://github.com/zalandoresearch/fashion-mnist`

[2] Apuntes curso de Redes Neuronales, clase 23 , *Dr. Tamarit, Francisco*, 2023. `https://www.famaf.unc.edu.ar/~ftamarit/redes2023/teoricos2023.html`

[3] Goodfellow I., Bengio Y., Courville A. , *Deep Learnig*, 2006.

[4] Hinton G., R. Salakhutdinov , *Reducing the Dimensionality of Data with Neural Networks*, 2006.

[5] Diederik P. Kingma, Jimmy Ba, *Adam: A Method for Stochastic Optimization*, 2017.