UNIVERSIDAD DE GRANADA

MNIST

RECONOCIMIENTO ÓPTICO DE CARACTERES

INTELIGENCIA COMPUTACIONAL

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

AUTOR

Pablo Valenzuela Álvarez (pvalenzuela@correo.ugr.es)

CONTENIDO

1. INTRODUCCIÓN

El objetivo de la primera práctica es resolver un problema de reconocimiento de patrones en dígitos manuscritos utilizando redes neuronales.

El conjunto usado será MNIST [1] (Modified National Institute of Standards Technology) creado por el Instituto Nacional de Estándares y Tecnología (NIST) estadounidense. Es un conjunto de datos de imágenes de dígitos escritos y se ha convertido en un estándar para en el ámbito del aprendizaje automático. Contiene 70000 imágenes de números del 0 al 9 en formato 28x28 que están etiquetados para facilitar la tarea del análisis de resultados una vez aplicados los algoritmos que necesitemos para nuestro problema.

Durante la realización de la práctica voy a explorar diversas técnicas sobre el conjunto de datos. Aplicaré algunos algoritmos basados en redes neuronales, buscando los parámetros óptimos para así mejorar la tasa de acierto. El objetivo de la práctica es aumentar esta tasa y reducir el error a menos de 1.

En resumen, he enfocado este documento como una memoria científica donde iré demostrando a través de los resultados obtenidos, porqué cada método que uso es mejor que el anterior hasta dar con el mejor.

2. REDES NEURONALES USADAS

2.1. PERCEPTRÓN

El perceptrón está inspirado en funcionamiento de las neuronas biológicas y es la unidad básica de procesamiento en una red neuronal. En esencia, es un modelo matemático que toma múltiples entradas, las suma y aplica una función de activación para producir una salida.

En general, esta red puede resolver problemas lineales, pero para problemas más desafiantes, se necesitan redes neuronales más profundas y complejas.

2.2. PERCEPTRÓN MULTI CAPA

Es una evolución de la red anterior, comprenden una red de múltiples capas de perceptrones interconectadas. Estas redes han demostrado ser eficaces en una gran variedad de aplicaciones, desde la clasificación de patrones hasta en el reconocimiento de voz y la toma de decisiones de forma autónoma. Tiene una gran capacidad para aprender a partir de datos, se pueden ajustar los pesos de las conexiones entre las capas, lo que permite la resolución de problemas complejos.

2.3. REDES CONVOLUCIONALES

Estas redes están diseñadas específicamente para procesar y analizar datos de tipo malla, como imágenes o vídeos, por lo que a priori, pueden ser la solución óptima a nuestro problema planteado en la práctica. Están formadas por varias capas entre las que destacan:

- **Convolucionales**: Aplican filtros para aprender patrones
- **Agrupación**: Se suelen aplicar después de la convolucionales, estas reducen la dimensionalidad y conservan las características más importantes.
- **Capas conectadas**: Son las que realizan la clasificación de las características extraídas.

3. CONFIGURACIÓN Y RESULTADOS

3.1. PERCEPTRÓN

Para el estudio de esta red se ha usado validación cruzada mostrada en la foto 1. También se ha preparado lista de algunos de los parámetros que esta red permite ajustar, para realizar una búsqueda que tiene como objetivo la obtención de la mejor combinación de estos. Esto se denomina búsqueda por hiperparámetros (ver foto 2).

```
crossV = RepeatedStratifiedKFold(
    n_splits=5,
    n_repeats=3,
    random_state=448
)
```

Foto 1. Configuración de la validación cruzada

```
hyperP = dict(
    max_iter=[5,10,30],
    eta0=[0.1,0.01,0.001]
)
```

Foto 2. Parámetros ajustables del Perceptrón

Los parámetros usados en la foto 2 significan.

1. **max_iter**: épocas o iteraciones que realizará el algoritmo antes de detenerse.
2. **eta0**: tasa de aprendizaje.

Parámetros	Media	Desviación típica
eta0: 0.01 max_iter: 10	**0.868383**	**0.009248**
eta0: 0.1 max_iter: 10	0.868306	0.009164
eta0: 0.001 max_iter: 10	0.868306	0.009164
eta0: 0.01 max_iter: 30	0.861817	0.023572
eta0: 0.1 max_iter: 30	0.861706	0.023520

Tabla 1. Mejores resultados de la búsqueda de hiperparámetros de Perceptrón.

Usando la mejor combinación de parámetros para este algoritmo sobre el conjunto de test obtenemos una precisión **de 0.879** y un error de **12.1.** Bastante lejos de ser la solución adecuada a nuestro problema.

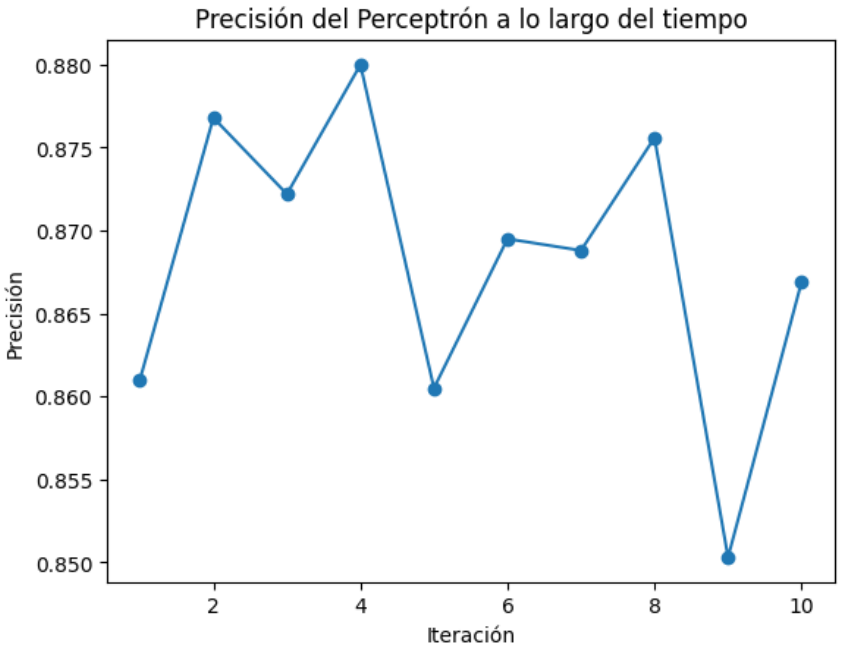

Foto 3. Evolución del acierto del Perceptrón

Se puede comprobar en la gráfica anterior (Foto 3) las dificultados que tiene este algoritmo para determinar la solución adecuada. Al ser un problema más complejo que los linealmente separables requiere una red más avanzada para su resolución.

3.2. PERCEPTRÓN MULTI CAPA

Para la siguiente red se usará la misma configuración para la validación cruzada vista en la foto 1. Los hiperparámetros sobre los que se buscará son:

- Las capas ocultas (**hidden layers**): son capas de neuronas que se sitúan entre la capa de entrada y la capa de salida.
- El tamaño del lote (**batch size**): la cantidad de ejemplos utilizados para actualizar los pesos del modelo.
- Las épocas (**maxi ter**): las repeticiones usadas en el entrenamiento del modelo.

```
hyperMLP = dict(
    batch_size=[256,512],
    hidden_layer_sizes=[(400,200), (800,400,200)],
    max_iter = [10,30],
)
```

Foto 4. Hiperparámetros usados en el Perceptrón Multicapa

Parámetros	Media	Desviación típica
batch_size: 256 hidden_layer_sizes: (800, 400, 200) max_iters: 30	**0.951733**	**0.007520**
batch_size: 256 hidden_layer_sizes: (400, 200) max_iters: 30	0.949883	0.006178
batch_size: 256 hidden_layer_sizes: (800, 400, 200) max_iters: 10	0.945817	0.003220
batch_size: 512 hidden_layer_sizes: (800, 400, 200) max_iters: 30	0.940383	0.005650
batch_size: 512 hidden_layer_sizes: (800, 400, 200) max_iters: 10	0.939233	0.005020

Tabla 2. Mejores resultados de la búsqueda de hiperparámetros de la red Perceptrón Multicapa

Los resultados obtenidos usando el mejor modelo de la tabla 2 sobre el conjunto de test da una precisión de **0.9758** y un error de **2.42**. Aunque es bastante mejor que el algoritmo anterior sigue sin ser la solución ideal para el problema. Quizá una búsqueda más amplia de hiperparámetros pueda reducir el error a valores cercanos a lo que se nos pide.

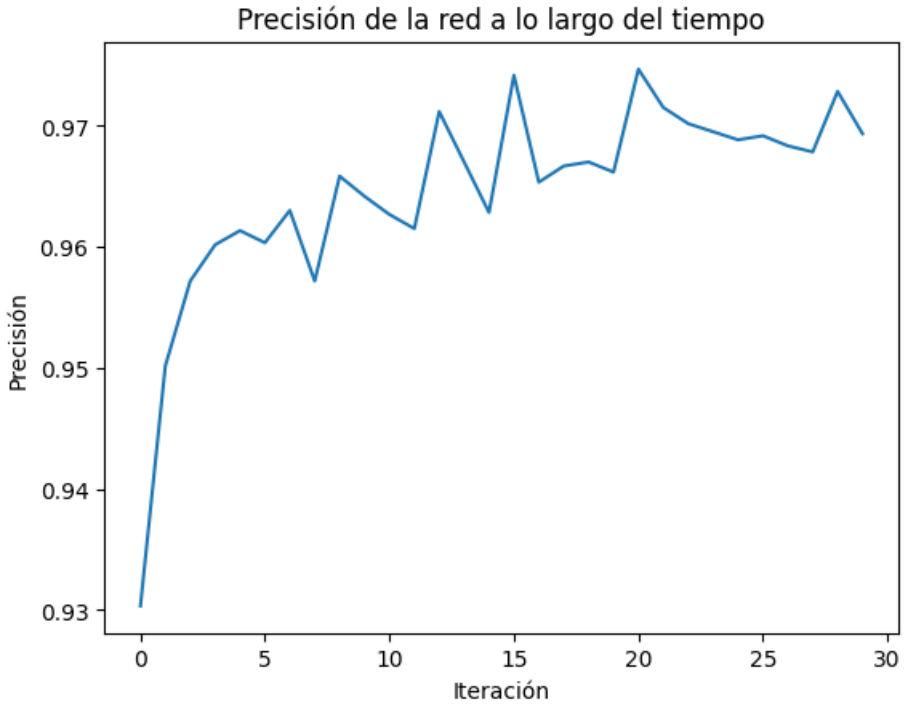

Foto 5. Evolución del acierto de Perceptrón Multicapa

Se puede observar en la gráfica de la foto 5 como el acierto va en constante crecimiento, aunque se ven bastantes altibajos al llegar a las 4-5 iteraciones y a partir de la iteración 20 empieza a bajar el acierto. El algoritmo utilizado es mejor capturando relaciones no lineales, pero la variabilidad en la escritura a mano y los distintos estilos hacen que siga siendo un problema complejo para esta red.

3.3. RED CONVOLUCIONAL

El último tipo de red que vamos a probar es la convolucional. El ejemplo visto en clase (ver foto 6) consta de dos capas de filtros 3x3 de tamaño 32 y 64 seguidos de capas de agrupación, una capa aplanadora y una capa de salida. Antes de la capa de salida se le añadió otra capa conectada con 200 neuronas.

Para el ejemplo no se utilizó validación cruzada.

```
capas = [
    keras.Input(shape=dimensiones),
    layers.Conv2D(32, kernel_size=(3,3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Conv2D(64, kernel_size=(3,3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Flatten(),
    layers.Dropout(0.5),
    layers.Dense(units=200, activation="relu"),
    layers.Dense(units=clases, activation="softmax"),
]
```

Foto 6. Configuración de la red convolucional inicial

Los resultados del ejemplo son de una precisión de ~ **0.992** y un error de ~ **0.801**. Lo cual ya nos serviría para concluir el estudio porque hemos bajado de 1 de error. Pero vamos a hacer una búsqueda para optimizar algunos parámetros y así ver como mejora el resultado.

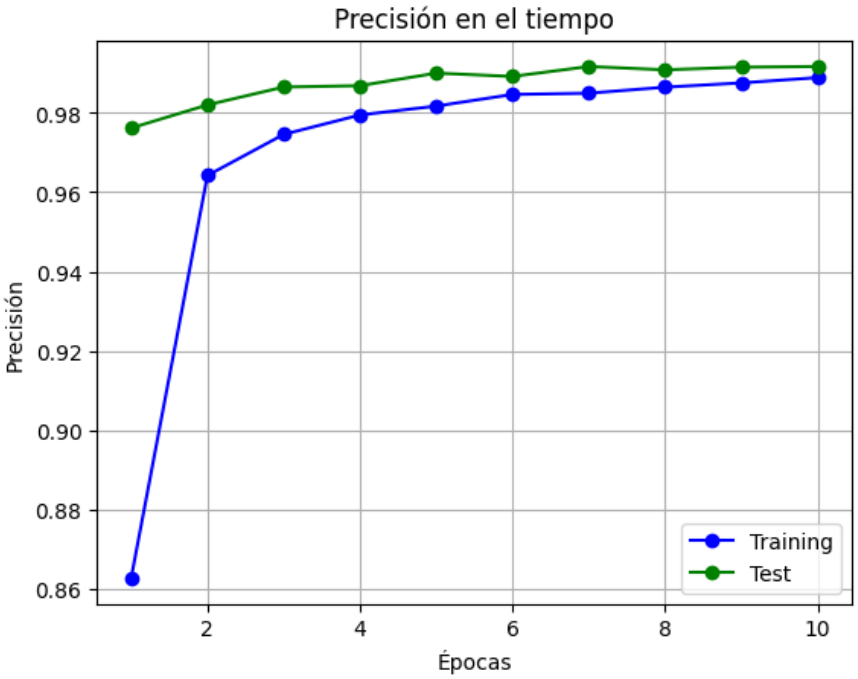

Foto 7. Evolución del acierto de la red convolucional inicial.

La gráfica de la foto 7 muestra el acierto sobre el conjunto de entrenamiento y test, y se puede apreciar la alta tasa de acierto de esta red aun teniendo poco ajustados los parámetros.

3.4. AJUSTE DE LOS PARÁMETROS DE LA RED CONVOLUCIONAL

EL algoritmo usado corresponde al de la foto 8, es una modificación del visto clase (foto 6). En esta versión se puede personalizar el número de capas convolucionales, el tamaño de los filtros, el ratio de sobre aprendizaje (dropout), el número de capas conectadas y el número de neuronas por capa conectada. El estudio de hiperparámetros se realizará sobre estas variables.

```python
def create_model(lay_Filters=2, kernel_size=(2,2), pool_size=(2,2), cUnits=4, units=200, rate=0.5):
    model = keras.Sequential()

    # añadimos las capas
    model.add(keras.Input(shape=dimensiones))

    # da errores si es mayor que 4
    if lay_Filters > 4:
        lay_Filters=4
    filters = 32

    # capas convolucionales y de agregacion que queramos
    for i in range(lay_Filters):
        model.add(layers.Conv2D(filters, kernel_size=kernel_size, activation="relu", padding="same"))
        model.add(layers.MaxPool2D(pool_size=pool_size))
        filters *= 2

    # capa que aplana
    model.add(layers.Flatten())

    # capa que ayuda a reducir el sobreajuste
    model.add(layers.Dropout(rate=rate))

    # for con las capas de neuronas que queramos añadir
    for i in range(cUnits):
        model.add(layers.Dense(units, activation="relu"))

    # capa de salida, el nº de nueronas tiene que coincidir con el nº de clases
    model.add(layers.Dense(clases, activation="softmax"))



    model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
    return model
```

Foto 8. Configuración del algoritmo de la red convolucional

```
hypers1 = dict(
    model__kernel_size=[(2,2),(3,3)],
    model__rate=[0.5,0.75],
    epochs=[10,30],
    batch_size=[512,256]
)

crossV = KFold(
    n_splits=3,
    shuffle=True,
    random_state=448
)
```

Foto 9. Configuración de inicial de hiperparámetros y validación cruzada.

Como muestra la foto 9, se usará una configuración inicial de hiperparámetros y validación cruzada.

Parámetros	Media	Desviación típica
batch_size: 512 epochs: 30 model__kernel_size: (3, 3) model__rate: 0.75	**0.9919833333333333**	**0.0003472111109332894**
batch_size: 256 epochs: 30 model__kernel_size: (3, 3) model__rate: 0.75	0.9918166666666667	0.0007597514213361051
batch_size: 512 epochs: 30 model__kernel_size: (3, 3) model__rate: 0.5	0.9912666666666666	0.0006637435917246917
batch_size: 256 epochs: 30 model__kernel_size: (3, 3) model__rate: 0.5	0.9912666666666666	0.0003965125751123501
batch_size: 256 epochs: 30 model__kernel_size: (2, 2) model__rate: 0.75	0.9906833333333332	0.0005661762583821269

Tabla 3. Mejores resultados de la primera búsqueda de hiperparámetros de mi red convolucional

Ahora, usando los valores de los parámetros del mejor modelo hacemos una segunda y última búsqueda con los valores de la foto 10.

```
hypers2 = dict(
    model__kernel_size=[(3,3)],
    model__rate=[0.75],
    epochs=[30],
    batch_size=[512],
    model__lay_Filters=[3,4],
    model__cUnits=[4,8],
    model__units=[400,800]

)

crossV = KFold(
    n_splits=3,
    shuffle=True,
    random_state=448
)
```

Foto 10. Segunda configuración de hiperparámetros de mi red convolucional.

Parámetros	Media	Desviación típica
'batch_size': 512 'epochs': 30 'model__cUnits': 4 'model__kernel_size': (3, 3) 'model__lay_Filters': 3 'model__rate': 0.75 'model__units': 800	**0.9933166666666667**	**0.0002392116682401473**
'batch_size': 512 'epochs': 30 'model__cUnits': 8 'model__kernel_size': (3, 3) 'model__lay_Filters': 3 'model__rate': 0.75 'model__units': 400	0.9928333333333333	0.0005948856099191798
'batch_size': 512 'epochs': 30 'model__cUnits': 8 'model__kernel_size': (3, 3) 'model__lay_Filters': 3 'model__rate': 0.75 'model__units': 800	0.9924333333333334	0.00010274023338285298
'batch_size': 512 'epochs': 30 'model__cUnits': 4 'model__kernel_size': (3, 3) 'model__lay_Filters': 3 'model__rate': 0.75 'model__units': 400	0.9918333333333335	0.0008106924338010181
'batch_size': 512 'epochs': 30 'model__cUnits': 4 'model__kernel_size': (3, 3) 'model__lay_Filters': 4 'model__rate': 0.75 'model__units': 800	0.9913	0.00024832774042921383

Tabla 4. Mejores resultados de la última búsqueda de hiperparámetros de mi red convolucional

Los resultados obtenidos con el mejor modelo de la tabla 4 sobre el conjunto de test son los siguientes:

- **Precisión ~ 0.9941**
- **Error ~ 0.5999**

Haciendo el estudio anterior hemos refinado unas décimas la tasa de acierto de la red convolucional y este resultado (al igual que el anterior) nos sirve para concluir el estudio del problema planteado para la práctica.

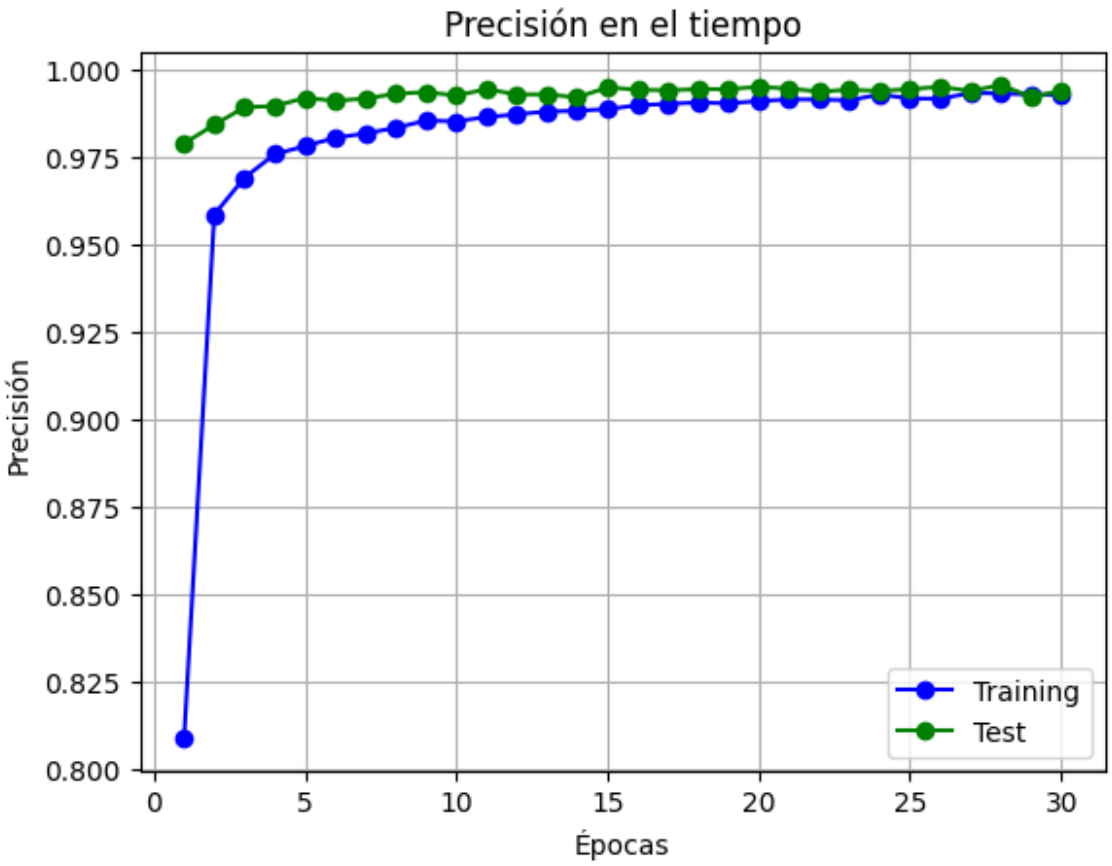

Foto 11. Evolución del acierto de mi red convolucional

En la última gráfica (foto 11) vemos la evolución de la tasa de acierto de la red convolucional que hemos diseñado. Podemos que la línea verde, correspondiente al conjunto de test; alcanza rápido un máximo local cerca de la onceaba época y a partir de ahí empieza a oscilar ligeramente.

4. CONCLUSIÓN

Tras haber realizado este estudio de búsqueda de hiperparámetros para distintas redes neuronales sobre el conjunto MNIST, los resultados obtenidos muestran una clara superioridad de las redes convolucionales sobre el resto de las redes utilizadas.

La capacidad de estas redes para extraer características y aprender de ellas ha demostrado ser esencial para la tarea encomendada en la práctica, la exploración exhaustiva de hiperparámetros realizada ha ayudado a aumentar de manera considerable el acierto y reducir por consiguiente el error hasta superar los límites fijados.

Por último, decir que, vistos los resultados las redes convolucionales son las ideales para realizar tareas de clasificación de imágenes ya que proporcionan una conclusión robusta y específica para este ámbito.

5. REFERENCIAS

[1] – The MNIST Database. http://yann.lecun.com/exdb/mnist/

[2] – Documentation Scikeras. https://adriangb.com/scikeras/stable/