Universidad Zaragoza

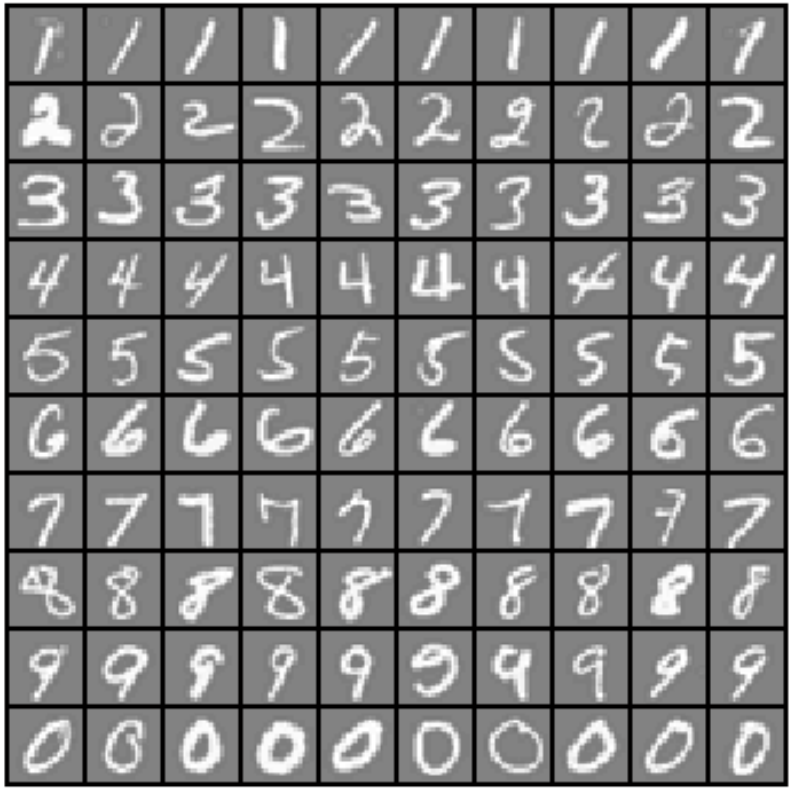
Inteligencia Artificial (30223)

Práctica P5: Redes Neuronales en Keras

Juan D. Tardós
Dpto. Informática e Ingeniería de Sistemas.

Reconocimiento de Dígitos Manuscritos

- Dataset MNIST: cada carácter 28x28 píxeles:



  - 60.000 muestras para entrenamiento y validación
  - 10.000 muestras para el test final

Objetivos de la práctica

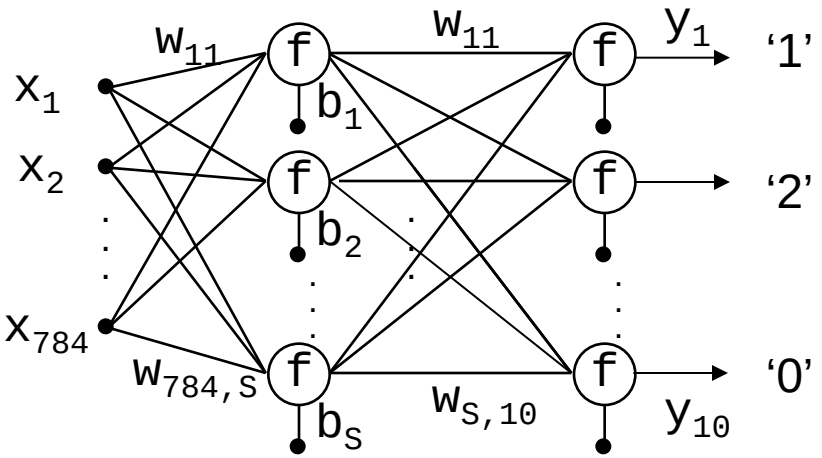
- Programar el entrenamiento y probarlo
  - Perceptrón
  - MLP: Multi-Layer Perceptron con una capa oculta (¿Nº neuronas?)
  - MLP: Multi-Layer Perceptron con dos capas ocultas (¿Nº neuronas?)
  - CNN: Convolutional Neural Network
- Evitar sobreajuste
  - Early Stopping, de las 60.000 muestreas de x_train
    - 90% para calcular los pesos
    - 10% para ver el error de validación y parar si no baja
    - Inconveniente: desaprovecha el 10% de los datos
  - Dropout:
    - Apagar aleatoriamente algunas neuronas al entrenar
    - Suele funcionar muy bien con redes complejas
- Análisis final de prestaciones: datos de test

MLP: Multi-Layer Perceptron

Entradas:
784 píxeles

1 o más
Capas ocultas

Salidas:
10 caracteres



- Funciones de activación
  - Capas ocultas: sigmoidal o RELU
    redes más sencillas redes más complejas
  - Capa de salida: sigmoidal o softmax
- La salida mayor → carácter reconocido
- Algoritmos de entrenamiento: SGD, RMSprop, Adam,...
- Función de coste: categorical_crossentropy, mean_squared_error,...

probar error cuadratico medio

Uso básico de Keras https://keras.io/

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()

model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5, batch_size=32)

loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)

classes = model.predict(x_test, batch_size=128)
```

10 neuronas d salida

capa de activacion

Two-Layer Perceptron

funcion de coste

Stochastic Gradient descent

entrena

epocas

calcular las metricas

calcular y obtenidas para esas x

Ejemplo: P5demo.py (1)

```
print('Loading MNIST dataset...')
# Problem dimensions
img_rows, img_cols = 28, 28
num_pixels = img_rows * img_cols
num_classes = 10

# The data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000, num_pixels)
x_test = x_test.reshape(10000, num_pixels)
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

las y cuando se carhan son cual es el digito de cad dato, la salida deseada que queremos que obtenga la red es todo 0, menos un 1 en el digito deseado

```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test  = keras.utils.to_categorical(y_test,  num_classes)

# Random permutation of training data
np.random.seed(0)
```

permutacion aleatoria

```
p = np.arange(x_train.shape[0])
```

se obtiene p, unu numero de 1 a 60000

```
np.random.shuffle(p)
```

se deseordenadn las x y las y con una p permutacion

porque yo no se como me van a llegar los datos, puede ser que primer
1, 2, por eso es importante mezclar todos para que a la hora de entrena

```
x_train = x_train[p]
y_train = y_train[p]
```

Ejemplo: P5demo.py (2)

```
# Función para parar cuando ya no mejora el error de validacion
earlystop=EarlyStopping(monitor='val_loss', patience=5,
                        verbose=1, mode='auto')
```

si el coste no disminuye despues de 5 iteraciones, parará

```
# Perceptron de un solo nivel
model = Sequential()
model.add(Dense(10, activation='sigmoid', input_shape=(num_pixels,)))
```

modelo denso con 10 neuras

```
model.compile(loss='categorical_crossentropy', optimizer=SGD(),
              metrics=['accuracy'])
model.summary()
```

esta funcion saca por pantalla la estructura ded la red que estamos probando

```
print('Training the NN....')
t0 = time.clock()
history = model.fit(x_train, y_train,
                    batch_size=128,
                    epochs=20,
                    validation_split=0.1,
                    #callbacks=[earlystop],
                    verbose=verbose)
train_time = time.clock() - t0
print('%s %.3f%s' %  ('Training time: ', train_time, 's') )
P5_utils.plot_history(history)
```

separa un 10% de los datos para comparar errores de entramiento con los de valida

¡Cuidado!
Entrenar con x_test => suspenso

funcion que nos da funciones extra, como el historial o una matriz de confusion

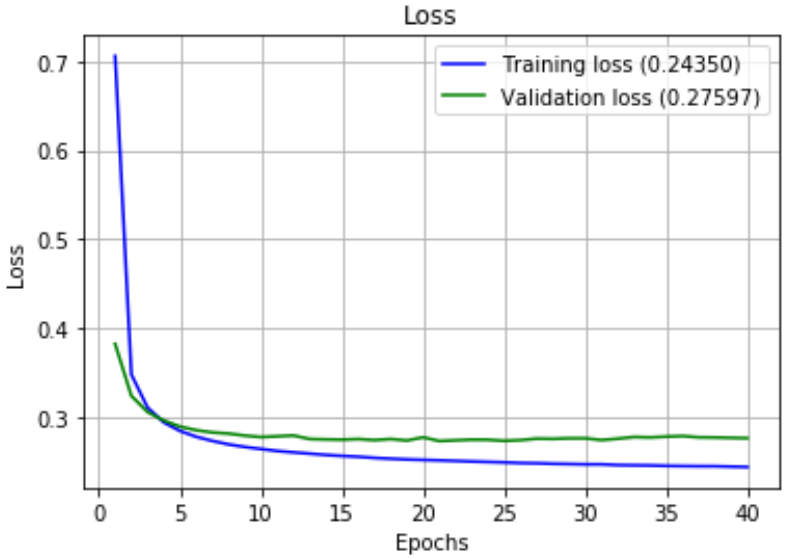
Ejemplo: P5demo.py (3)

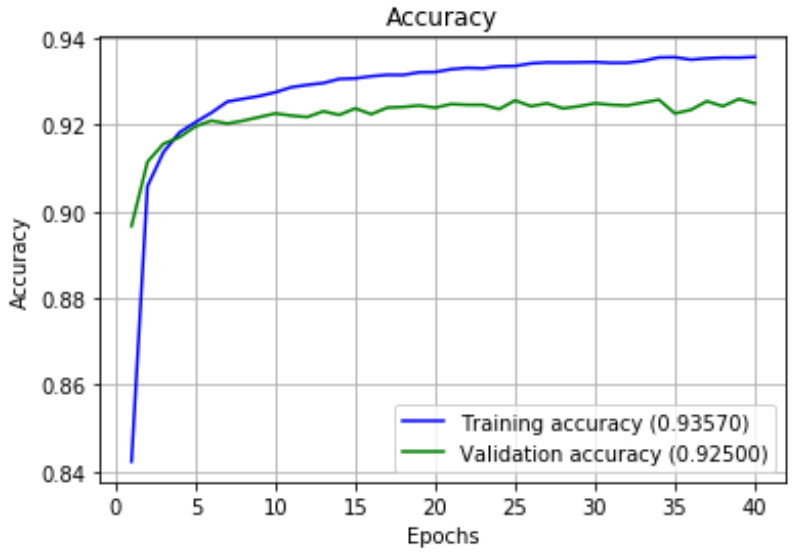
```
# Evaluar la red
train_score = model.evaluate(x_train, y_train, verbose=0)
test_score = model.evaluate(x_test, y_test, verbose=0)
print('%s %2.2f%s' % ('Accuracy train: ', 100*train_score[1], '%' ))
print('%s %2.2f%s' % ('Accuracy test:  ', 100*test_score[1], '%'))

y_pred = model.predict(x_test)
P5_utils.plot_mnist_confusion_matrix(y_test, y_pred)
```

Aquí es donde usamos x_test

Construimos la matriz de confusion

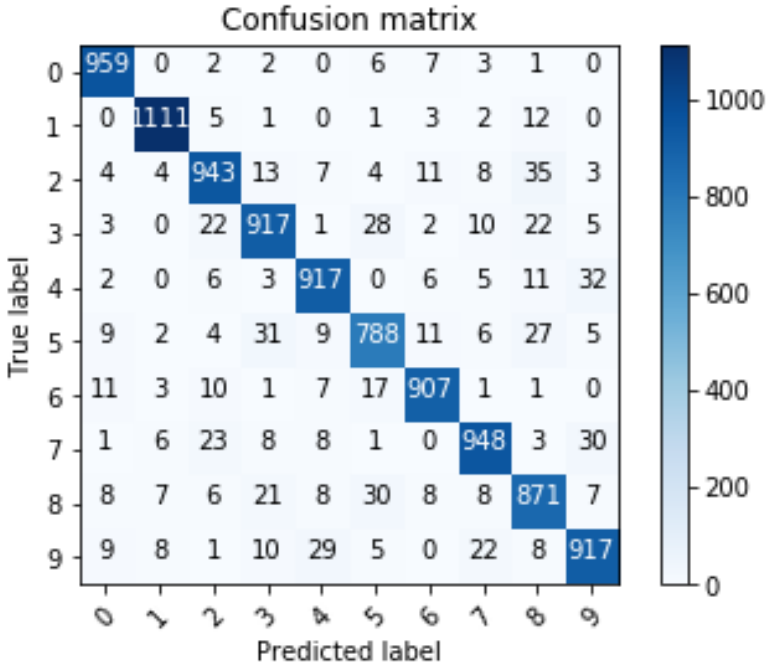
Ejemplo de resultado





- Accuracy train: 93.54%
- Accuracy test: 92.78%

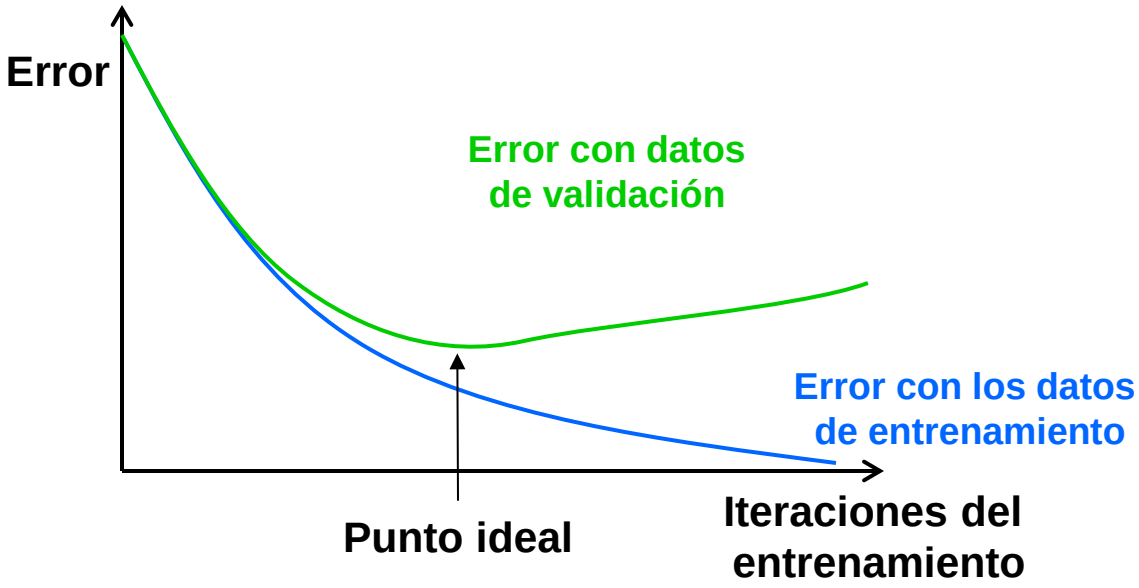
Ve anotando en una tabla los resultados obtenidos con cada red que pruebes



Universidad Zaragoza
1542

Como evitar el sobreajuste (1)

- Parada del entrenamiento (early stopping)




```
earlystop=EarlyStopping(monitor='val_loss', patience=5,
                        verbose=1, mode='auto')
history = model.fit(x_train, y_train, batch_size=128, epochs=20,
                    validation_split=0.1,
                    callbacks=[earlystop],
                    verbose=True)
```

Como evitar el sobreajuste (2)

- Dropout **tecnica para evitar el sobreajuste**

```
model = Sequential()
        model.add(Dense(512, activation='relu',
                        input_shape=(num_pixels,)))
        model.add(Dropout(0.2))
        model.add(Dense(num_classes, activation='softmax'))
```

apago el 20%

añado capa de salida

- Regularización

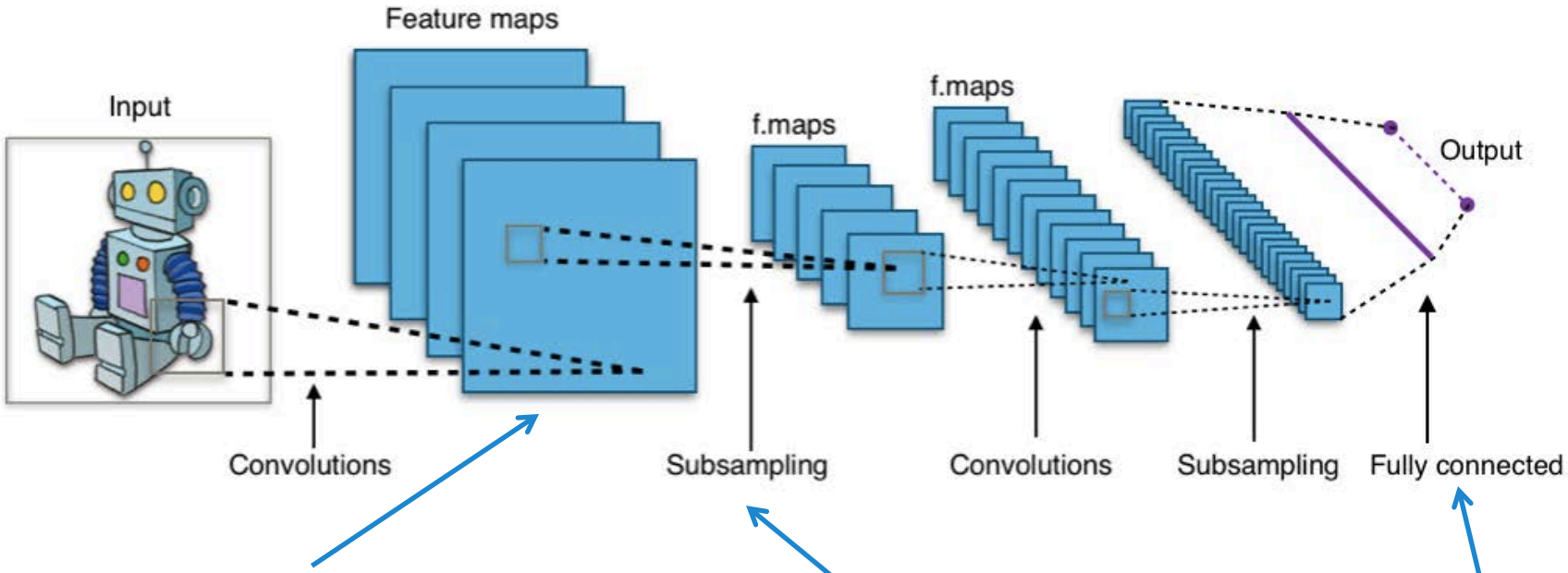
```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01)))
```

añado una capa densa

coeficiente 0.01

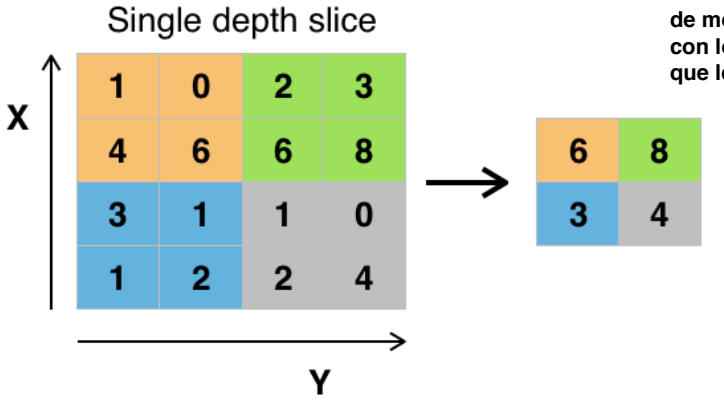
  - Añade al coste 0.01 por la norma L2 de la matriz de pesos
    - Norma L2 → favorece que los pesos sean pequeños
    - Norma L1 → favorece que algunos pesos sean cero

CNN: Red Neuronal Convolucional



Convolutional Layer
Ej: 4 filtros con Kernel 5x5

Subsampling layer
Ej: Max-Pooling 2x2

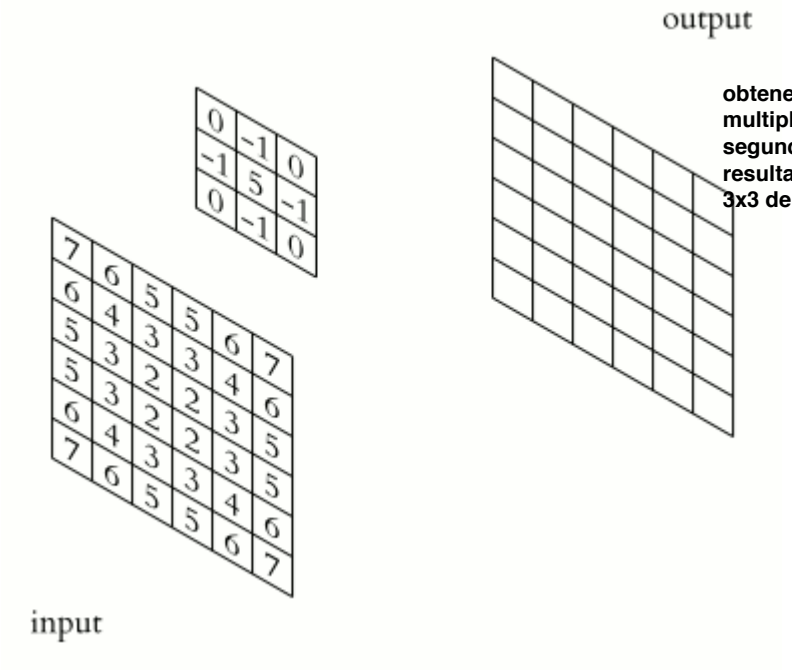


Multi-Layer Perceptron

de modo que el perceptron ya no trabaja con los datos iniciales sino con las datos que les devuelve las capas convolucionales

Convolución

- Aplicar un Kernel de tamaño nxn (matriz de pesos fija) a cada grupo de nxn pixeles vecinos de una imagen:



obtenemos cada cuadradito de esta matriz multiplicando un cuadro 3x3 del primero con el segundo cuadrado 3x3 , la posicion del cuadradito resultado es la misma que la del centro del cuadrado 3x3 de la primera matriz

  - Cada pixel de salida depende de nxn pixeles de entrada
  - Es una operación local

Ejemplo: mnist_cnn.py

2 capas convolucionales + las 2 capas del perceptron

```
# input image dimensions
img_rows, img_cols = 28, 28

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),          # 32 filtros con kernel 3x3
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))  # 64 filtros con kernel 3x3
model.add(MaxPooling2D(pool_size=(2, 2)))         # Submuestreo
model.add(Dropout(0.25))                          # Para reducir sobreajuste
model.add(Flatten())                              # Convertir a un vector
model.add(Dense(128, activation='relu'))          # Perceptrón de dos capas
model.add(Dropout(0.5))                           # Para reducir sobreajuste
model.add(Dense(num_classes, activation='softmax')) # Salida softmax
```