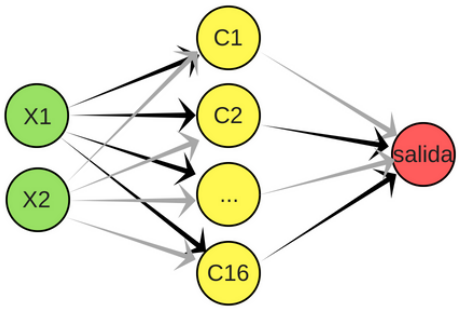
03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

1 PRIMERA RED NEURONAL

Dada la función XOR:

x1	x2	x1 XOR x2
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

En la que tenemos 2 entradas y 1 salida. Podemos hacer una Red Neuronal con una primera capa que representa las entradas (x1,x2), con tantas neuronas como entradas tenemos, una capa intermedia de 16 neuronas y una capa de salida con una neurona que nos dará el resultado de la operación.



Para realizar esta disposición, creamos un modelo de red neuronal secuencial (una capa delante de otra) con 2 capas, Pero en realidad son 3 capas ya que en la primera capa le decimos cuantas entradas tienen (que se refieren a la primera capa)

```
model = keras.Sequential([
    keras.layers.Dense(16,input_shape=(2,), activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
```

Una vez, creada se ajusta el modelo indicando la función de pérdida, optimización y la métrica:

```
model.compile(loss='mean_squared_error',
              optimizer='adam',
              metrics=['binary_accuracy'])
```

03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

Si temenos los siguientes datos:

```
training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
```

y estos son los resultados que se obtienen, en el mismo orden

```
target_data = np.array([[0],[1],[1],[0]], "float32")
```

Se entrena el modelo de la siguiente manera:

```
model.fit(training_data, target_data, epochs=1000)
```

Indicamos con model.fit() las entradas y sus salidas y la cantidad de iteraciones de aprendizaje (epochs) de entrenamiento.

```
Epoch 256/1000
1/1 [==============================] - 0s 16ms/step - loss: 0.1824 - binary_accuracy: 0.7500
Epoch 257/1000
1/1 [==============================] - 0s 23ms/step - loss: 0.1821 - binary_accuracy: 1.0000
Epoch 258/1000
1/1 [==============================] - 0s 21ms/step - loss: 0.1818 - binary_accuracy: 1.0000
Epoch 259/1000
1/1 [==============================] - 0s 20ms/step - loss: 0.1815 - binary_accuracy: 1.0000
```

Vemos que a partir de la iteración 257 ya va perfecta.

Una vez entrenada, podemos evaluar y predecir:

Evaluamos:

```
scores = model.evaluate(training_data, target_data)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
1/1 [==============================] - 0s 262ms/step - loss: 0.0342 - binary_accuracy: 1.0000
binary_accuracy: 100.00%
```

Hacemos las 4 predicciones posibles de XOR, pasando nuestras entradas:

```
print (model.predict(training_data).round())
```

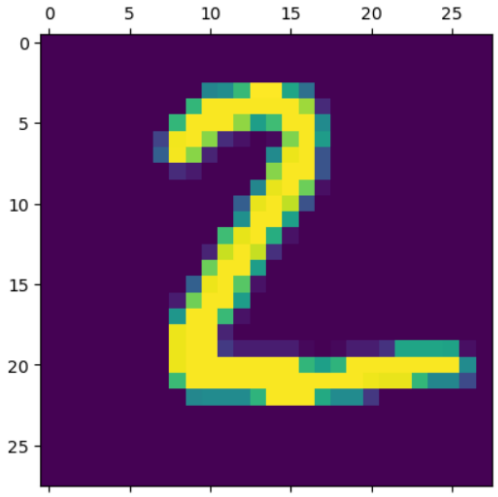
```
1/1 [==============================] - 0s 132ms/step
[[0.]
 [1.]
 [1.]
 [0.]]
```

Vemos que el resultado para las 4 entradas es el correcto.

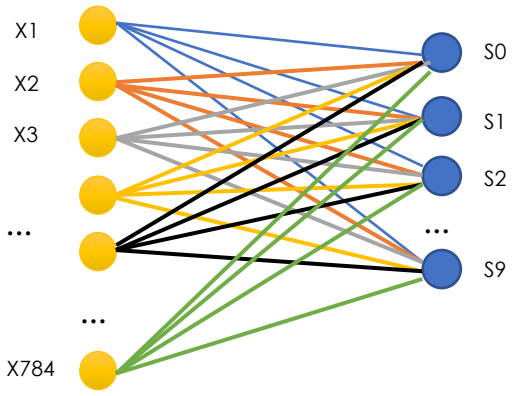
03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

2 Un ejemplo más elaborado

El framework keras incorpora datasets de ejemplo, y uno que ya hemos usado anteriormente (digits) lo tiene incorporado con. Más precisión. En lugar de números de 8x8, tiene una matriz de 28x28 valores de escala de grises entre 0 y 255. El total de datos es de 60000 muestras.



Se quiere crear una red neuronal con 28x28 = 784 entradas (los valores de cada pixel entre 0 y 255) y una capa de salida de 10 salidas (la probabilidad de que la entrada sea un 0, un 1 ,… un 9) Estas salidas estarán entre 0 y 1



2.1 Creación de la RN

El dataset se carga de la siguiente manera:

```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()
```

03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

Cada muestra de entrenamiento y test son arrays de 28x28 cada uno, pero como la Red Neuronal necesita valores de 1 dimensión, se pasan a vectores:

```
X_train_lista = X_train.reshape(len(X_train),28*28)
X_test_lista = X_test.reshape(len(X_test),28*28)
```

Ahora se puede generar una red neuronal con solamente una capa de salida de 10 nodos y la entrada de 784 nodos. Sería de la siguiente manera:

```
model = keras.Sequential([
    keras.layers.Dense(10,input_shape=(784,), activation='sigmoid')
])
```

El parámetro input_shape indica cuántas entradas tiene la red neuronal. Dense nos indica que los nodos están todos conectados (es una red "densa") y la función de activación es sigmoide (Las funciones de activación se verán más adelante)

El siguiente paso es establecer los parámetros de la red y entrenarla.
El primero de estos argumentos es la función *de pérdida (loss)* que usaremos para **evaluar el grado de error entre las salidas calculadas y las salidas deseadas de los datos de entrenamiento.**
Por otro lado, especificamos un **optimizador** que, como veremos, es la **forma en que tenemos que especificar el algoritmo de optimización que permite a la red neuronal en Keras calcular el peso de los parámetros de los datos de entrada y la función de pérdida definida.**

Y finalmente debemos indicar la **métrica** que usaremos para monitorear el proceso de aprendizaje (y prueba) de nuestra red neuronal.

En este ejemplo solo consideraremos la precisión (fracción de imágenes que están clasificadas correctamente).

```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Una vez que nuestro modelo ha sido definido y el método de aprendizaje configurado, está listo para ser entrenado. Para esto podemos entrenar o "ajustar" el modelo a los datos de entrenamiento disponibles invocando el método *fit ()* del modelo

```
model.fit(X_train_lista, y_train, epochs=5)
```

Con *epochs* indicamos el número de veces que utilizaremos todos los datos en el aprendizaje proceso.

03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

2.2 Evaluación del modelo

En este punto, la red neuronal en Keras se ha entrenado y su comportamiento con los nuevos datos de prueba ahora se puede evaluar utilizando el método de evaluate(). Este método devuelve dos valores:

```
test_loss, test_acc = model.evaluate(X_test_lista, y_test)
```

Estos valores indican qué tan bien o mal se comporta nuestro modelo con los nuevos datos que nunca ha visto.

Estos datos han sido almacenados en *x_test_lista* y *y_test* cuando hemos realizado *mnist.load_data ()* y los pasamos al método como argumentos.

La salida del evaluate nos indica la pérdida y la exactitud.

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2690 -
accuracy: 0.9251
```

Aplicado a datos que el modelo nunca había visto antes, clasifica el 92% de ellos correctamente.

En Machine Learning, una herramienta muy útil para evaluar modelos es **la matriz de confusión**. Una tabla con filas y columnas que cuentan las predicciones en comparación con los valores reales.

Utilizamos esta tabla para comprender mejor cómo se comporta el modelo y es muy útil mostrarlo explícitamente cuando una clase se confunde con otra.

		Predicted class	
		positive	negative
Actual class	positive	TP	FN
	negative	FP	TN

Verdaderos positivos (TP), verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN), son los cuatro resultados posibles diferentes de una predicción única para un caso de dos clases con clases "1" ("positivo") y "0" ("negativo").

Un falso positivo es cuando el resultado es incorrectamente clasificado como positivo, cuando de hecho es negativo.

03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

Un falso negativo es cuando el resultado se clasifica incorrectamente como negativo cuando de hecho es positivo. Verdaderos positivos y verdaderos negativos son obviamente clasificaciones correctas.

Con esta matriz de confusión, la **exactitud** se puede **calcular sumando los valores de la diagonal y dividiéndolos por el total:**

Precisión = (TP + TN) / (TP + FP + FN + TN)

Sin embargo, la precisión puede ser engañosa en términos de la calidad del modelo porque, al medirlo para el modelo concreto, **no distinguimos entre falso positivo y errores de tipo falso negativo, como si ambos tuvieran la misma importancia.**

Por ejemplo, en un modelo que prediga si un hongo es venenoso el costo de un falso negativo, es decir, un hongo venenoso dado para el consumo podría ser dramático. Por el contrario, un falso positivo tiene un costo muy diferente.

Por esta razón tenemos otra métrica llamada ***Sensibilidad*** (o ***recuerdo***) que nos dice qué tan bien el modelo evita las falsas negativos:

Sensibilidad = TP / (TP + FN)

En otras palabras, del total de observaciones positivas (hongos venenosos), cuántos detecta el modelo.

De la matriz de confusión se pueden obtener varias métricas para enfocar otros casos.

La conveniencia de usar una métrica u otra dependerá de cada caso particular y, en particular, del "costo" asociado con cada error de clasificación del modelo.

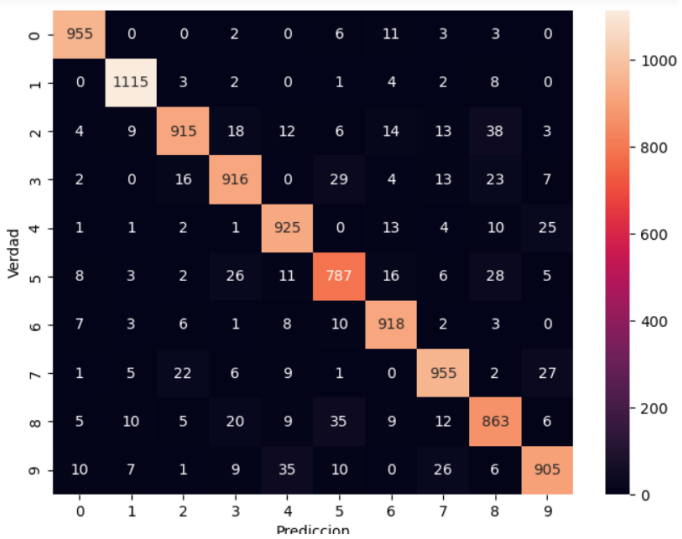
03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

Pero ¿cómo creamos la matriz de confusión para el ejemplo de los dígitos manuscritos?

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 955,    0,    0,    2,    0,    6,   11,    3,    3,    0],
       [   0, 1115,    3,    2,    0,    1,    4,    2,    8,    0],
       [   4,    9,  915,   18,   12,    6,   14,   13,   38,    3],
       [   2,    0,   16,  916,    0,   29,    4,   13,   23,    7],
       [   1,    1,    2,    1,  925,    0,   13,    4,   10,   25],
       [   8,    3,    2,   26,   11,  787,   16,    6,   28,    5],
       [   7,    3,    6,    1,    8,   10,  918,    2,    3,    0],
       [   1,    5,   22,    6,    9,    1,    0,  955,    2,   27],
       [   5,   10,    5,   20,    9,   35,    9,   12,  863,    6],
       [  10,    7,    1,    9,   35,   10,    0,   26,    6,  905]],
      dtype=int32)>
```

Pero la podemos ver en un heatmap:

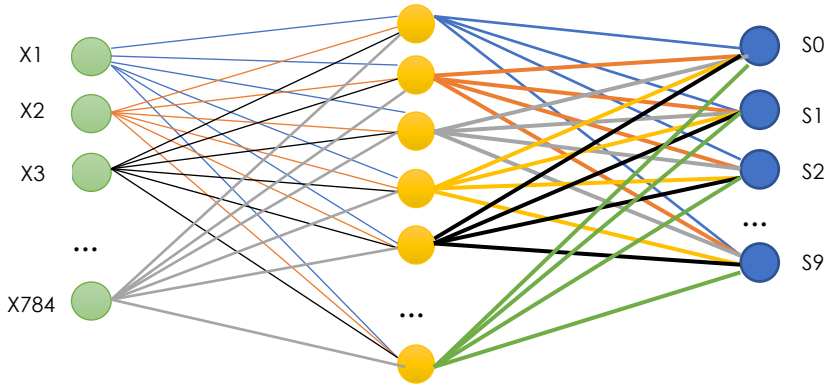
```
import seaborn as sn
plt.figure(figsize = (10,10))
sn.heatmap(cm, annot = True, fmt = 'd')
plt.xlabel('Prediccion')
plt.ylabel('Verdad')
```



2.3 Mejora del modelo

El modelo de red neuronal se puede mejorar añadiendo una capa intermedia de neuronas, de esta manera:

03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza




```
model = keras.Sequential([
    keras.layers.Dense(100,input_shape=(784,), activation='relu'),
    keras.layers.Dense(10, activation='sigmoid')
])
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
model.fit(X_train_lista, y_train, epochs=5)
```

La evaluación del modelo sería:

```
model.evaluate(X_test_lista, y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0765 - accuracy: 0
.9768
```

Y la matriz de confusión:

```
tf.math.confusion_matrix(labels= y_test, predictions = y_predict_label)
```

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 971,    0,    1,    2,    0,    0,    2,    1,    2,    1],
       [   0, 1118,    5,    0,    0,    2,    2,    1,    7,    0],
       [   1,    1, 1015,    2,    2,    0,    1,    4,    5,    1],
       [   0,    0,    8,  987,    0,    7,    0,    3,    3,    2],
       [   0,    0,    8,    0,  940,    1,    2,    3,    6,   22],
       [   2,    0,    0,    9,    1,  874,    2,    1,    2,    1],
       [   7,    2,    2,    1,    1,    7,  935,    1,    2,    0],
       [   1,    5,   10,    2,    0,    0,    0, 1000,    4,    6],
       [   4,    0,    2,    5,    2,    4,    1,    6,  945,    5],
       [   3,    4,    0,    4,    4,    5,    0,    0,    6,  983]],
      dtype=int32)>
```

03008915 C/ Ferrocaril, 22, 03570 La Vila Joiosa **Tel** 966870140 **Fax** 966870141 http://portal.edu.gva.es/iesmarcoszaragoza

Con el heatmap asociado: