

Entrenamiento y clasificación en Keras

Visión por Computador, curso 2024-2025

Silvia Martín Suazo, silvia.martin@u-tad.com

6 de noviembre de 2024

U-tad | Centro Universitario de Tecnología y Arte Digital



Tipos de clasificación

Problemas de clasificación

Para que una red neuronal artificial sea capaz de realizar la clasificación esta tiene que lograr diferentes objetivos:

- **Extracción de características:** Diferenciar en la imagen distintas formas, texturas, objetos, etc.
- **Clasificación de características:** Los elementos anteriormente identificados tienen que ser analizados para obtener una salida con sentido.
- **Generalización del problema:** La red debe ser robusta ante imágenes con las que no ha sido entrenada.
- **Invariante ante cambios:** Es deseable que la red sea capaz de clasificar ante variabilidad en la composición de sus imágenes.
- **Estabilidad:** En general el comportamiento de las redes neuronales es delicado.

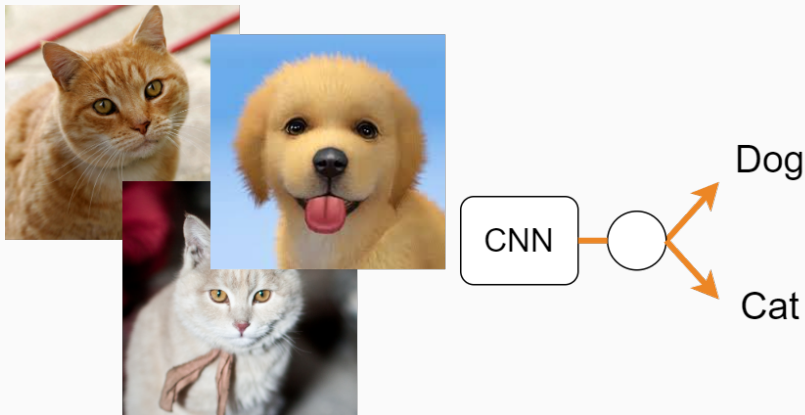
Uno de los problemas más básicos que tratan las redes neuronales es la clasificación. Dentro de ella encontramos distintos tipos de problemas:

- Clasificación binaria
- Clasificación multi-clase
- Clasificación multi-etiqueta
- Clasificación jerárquica

Clasificación binaria

Uno de los problemas más básicos que tratan las redes neuronales es la clasificación. Dentro de ella encontramos distintos tipos de problemas:

- Clasificación binaria



Entrenamiento en Keras

Carga de imágenes

A la hora de realizar un entrenamiento con imágenes haciendo uso de las librerías `tensorflow/keras` es necesario `cargar las imágenes` de las que se va a hacer uso.

Para ello, existen dos alternativas principales:

- Carga de datos “a mano” usando `numpy`.
- Carga haciendo uso de la librería `keras`.

**Es muy importante que todas las imágenes tengan el mismo tamaño y este sea indicado en la capa de entrada*

Carga de imágenes “a mano”

Una de las aproximaciones para cargar los datos y realizar un **entrenamiento** con ellos es usar la librería **numpy** para generar una matriz de imágenes.

El proceso de carga se puede dividir en los siguientes pasos:

- Recorrer la estructura de **carpetas** para acceder a **todas las imágenes** del dataset.
- Cada imagen que se encuentra se **abre** y procesa.
 - Si es necesario, se realiza una transformación de **color** a **blanco y negro**.
 - Si es necesario, se modifican las **dimensiones** de la imagen.
- Por cada imagen procesada, se guarda una **etiqueta** que identifique su clase.
- Una vez cargadas todas las imágenes, se realiza una **normalización** de sus píxeles.

** este proceso puede ser modificado y dependerá del problema y la estructura del dataset.*

La librería `keras` proporciona funciones para realizar la carga de datos de manera `automática` y `eficiente`.

La función `keras.utils.image_dataset_from_directory` está diseñada para cargar datos desde un directorio. El uso de esta librería proporciona una mayor `facilidad` y `eficiencia`, pero a cambio se pierde cierta `flexibilidad`.

El siguiente [artículo de la API de Keras](#) explica el funcionamiento y parámetros de la función.


El etiquetado es el proceso de **asignación de categorías** al conjunto de datos. De esta forma se puede identificar cada categoría mediante términos específicos.

Estos términos pueden ser de la siguiente forma:

- Codificación One-Hot.
- Codificación con valores numéricos.

Etiquetado One-Hot

En el etiquetado **One-Hot** a cada categoría se le asigna un **vector de valores binarios** donde cada posición corresponde a una clase. Se recomienda su utilización con un número bajo de clases, como en la clasificación binaria.

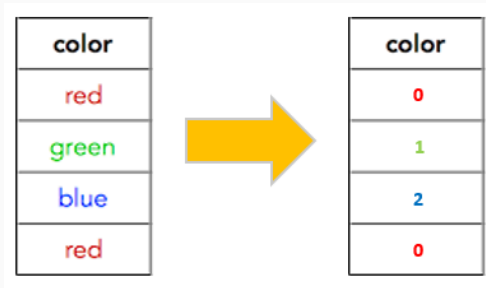


color
red
green
blue
red

color_red	color_blue	color_green
1	0	0
0	0	1
0	1	0
1	0	0

Etiquetado con numeración

En el etiquetado con **numeración** a cada categoría se le asigna un **valor numérico** de 1 a n , siendo n el número de clases.



Implementación del modelo

Para definir un modelo, en este caso una CNN, en primer lugar es necesario generar un modelo secuencial.

```
model = models.Sequential()
```

A continuación se le van añadiendo las capas, las cuales han sido explicadas durante las presentaciones anteriores.

```
Ex: model.add(Dense(units=1))
```

**Generalmente se implementa en una función para probar con distintos hiperparámetros*

Compilación del modelo

En Keras la compilación del modelo consiste en la **configuración** del modelo para su entrenamiento. En esta función se indica:

- La **función de pérdida**
- El **optimizador**
- Una lista de **métricas** para la evaluación del modelo.

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

Optimizadores

A la hora de realizar el **entrenamiento** de una red convolucional, es necesario definir un **optimizador** que ejecute de manera **eficiente** el algoritmo de descenso del gradiente.

La elección de optimizador puede tener **drásticas** consecuencias en diferentes aspectos del entrenamiento como la velocidad de convergencia, la estabilidad, etc.

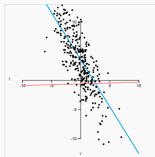
1. Generate a dataset

Select a training set size n .

☐ Small ☒ Medium ☐ Large

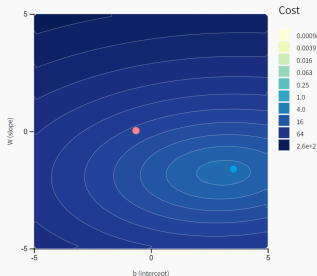
A training set of the chosen size will be sampled with noise from a line representing ground truth. This line is the target line for the model defined by $y = wx + b$.

Generate a new dataset.



2. Observe the cost landscape and initialize parameters.

The cost function is the L2 loss (defined as $\mathcal{L}(y, \hat{y}) = \|y - \hat{y}\|^2$) averaged over the training set. The blue dot indicates the value of the cost function at the ground truth slope and intercept. The red dot indicates the value of the cost function at a chosen initialization of the slope and intercept. Drag and drop the red dot to change the initialization.



3. Optimize the cost function

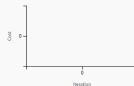
Now you can update the parameters iteratively to minimize the cost. Select a learning rate.

☐ Small ☒ Medium ☐ Large

Select a batch size.

☒ $n_b = 1$ ☐ $n_b = 24$ ☐ $n_b = \infty$

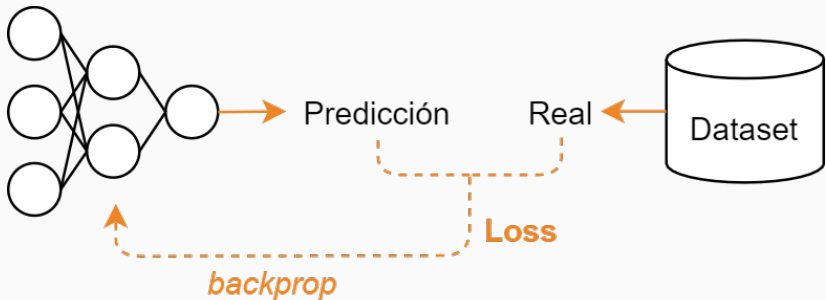
Train the model.



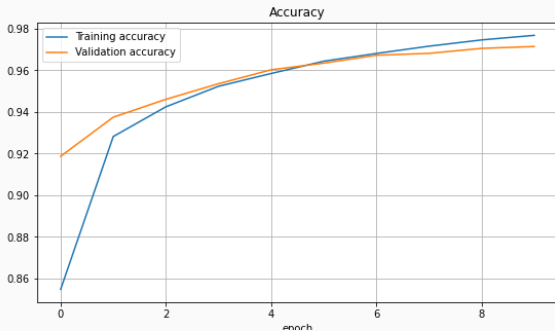
Funciones de pérdida

Dependiendo del **problema** que se vaya a tratar se definirá una **función de pérdida** que se adapte al mismo. Esta medida servirá para calcular la **distancia** entre la **predicción** de las redes neuronales y la **salida real**.

La definición de una función de pérdida **adecuada** es un paso **crucial** para el funcionamiento correcto de las redes neuronales.



La **exactitud** o **accuracy** (a veces mal traducida como **precisión**) da la **tasa de aciertos** de la red a la hora de **realizar sus predicciones**.



Métricas

La **matriz de confusión** es una de las **mejores** medidas para evaluar el **desempeño** de un clasificador (binario o no). Esta **agrupa** las predicciones de la red en una tabla.

<i>Real label</i>	0	True Negative	False Positive
	1	False Negative	True Positive
		0	1
		<i>Predicted label</i>	

Métricas

La **matriz de confusión** es una de las **mejores** medidas para evaluar el **desempeño** de un clasificador (binario o no). Esta **agrupa** las predicciones de la red en una tabla.

Real label

0				
1				
2				
3				
	0	1	2	3

Predicted label

**Para el caso multiclase*

A partir de la matriz de confusión se pueden obtener distintas métricas útiles:

- **Accuracy:** $\frac{TP+TN}{TP+TN+FP+FN}$

- **Recall:** $\frac{TP}{TP+FN}$

- **Precision:** $\frac{TP}{TP+FP}$

A partir de la matriz de confusión se pueden obtener distintas métricas útiles:

- **Specificity:** $\frac{TN}{TN+FP}$
- **Negative predictive value:** $\frac{TN}{TN+TP}$
- **F1-Score:** $\frac{2*Recall*Precision}{Recall+Precision}$

Las métricas además de indicarse durante la compilación del modelo, pueden calcularse de forma normal a partir de los valores predichos y los reales.

Para ello, Keras y sklearn ofrecen funciones en `keras.metrics` y `sklearn.metrics`:

```
Ex: metrics.MeanSquaredError().update_state(y_real,  
                                              y_pred)
```

```
Ex: metrics.f1_score(y_real, y_pred,  
                     average='binary', zero_division=0)
```

El entrenamiento se realiza con la función `fit`:

```
model.fit(X_train, y_train, batch_size=32,  
epochs=100, callbacks=[callback], verbose=0,  
validation_data = (X_val, y_val))
```

Donde `X_train` y `X_val` son las **particiones de entrenamiento y validación** respectivamente, y `y_train` y `y_val` las etiquetas.

`Batch_size` indica el tamaño de las particiones de datos, llamados **lotes**, que se entrenarán cada vez. Los lotes se utilizan para aligerar la carga de memoria y aumentar la generalización.

El entrenamiento se realiza con la función *fit*:

```
model.fit(X_train, y_train, batch_size=32,  
epochs=100, callbacks=[callback], verbose=0,  
validation_data = (X_val, y_val))
```

Los *epochs* son el número de veces que se recorren los datos de entrenamiento durante esta fase.

Verbose es un parámetro que permite controlar la cantidad de información que se muestra sobre el entrenamiento. 0 indica ninguna información, 1 información detallada y 2 información resumida.

Keras callbacks

Los **callbacks** de Keras son herramientas que permiten **customizar** el comportamiento del entreno de un modelo. Estas se introducen dentro de la función **fit**.

Los callbacks permiten realizar acciones:

- En cada:
 - **Epoch**
 - **Batch**
 - **Entreno**
 - **Test**
 - **Evaluación**
- Durante los momentos:
 - **Al comienzo de**
 - **Al final de**

* *para más información acudir al a [guía oficial de Keras](#)*

Learning Rate Scheduler

Modifica el valor del **learning rate** durante el entrenamiento.

Debe definir una **política de actualización** que recibe el **epoch** y **learning rate** actuales.

Model checkpoint

Permite **guardar** los pesos de la red neuronal durante su entrenamiento.

Se puede definir cierta **lógica** para guardar los pesos sólo cuando sera interesante.

Early Stopping

Termina el entrenamiento cuando ciertas condiciones se cumplen.

Se utiliza como mecanismo de **regularización** para terminar un entrenamiento antes de que suceda overfitting. Existen distintos parámetros para su configuración, pero por ejemplo, se puede especificar que si la **precisión** de un clasificador no mejora durante X epochs, se termine el entrenamiento.

Tensorboard

Tensorboard es una herramienta que permite **monitorizar** un entrenamiento en tiempo real.

Custom callbacks

También es posible definir callbacks de manera **personalizada**.

Para realizar las pruebas o utilizar el modelo de forma normal, se utiliza *evaluate* o *predict*, dependiendo de si queremos evaluar directamente métricas o no:

model.evaluate(X, y) model.predict(X)

Donde X son los datos sobre los que se quiere predecir o clasificar.

Se denomina **ajuste de hiperparámetros** a la búsqueda de aquellos valores para los hiperparámetros del modelo, con los que se obtiene un mayor rendimiento. Es decir, se **buscan los hiperparámetros óptimos**.

Esta búsqueda puede realizarse **mediante bucles** o mediante **librerías como Optuna**.

Tipos de clasificación

Clasificación binaria: capa completamente conectada

En una clasificación binaria, la capa de salida generalmente consta de **una sola neurona** con una función de activación sigmoide. Esta neurona produce **una probabilidad** que representa la clase positiva o negativa. Es importante que las etiquetas estén en codificación one-hot o numérica, representando una sola clase, de tal forma que la clasificación sea si **pertenece o no**.

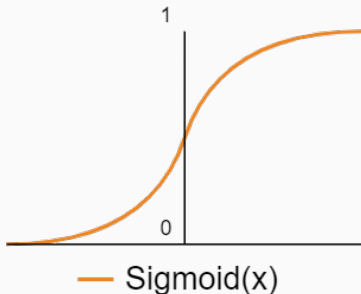
Ex: `layers.Dense(1,activation='sigmoid')`

Clasificación binaria: activación de capa de salida

Dentro de los problemas de **clasificación** la salida de la red debe ser escogida con especialmente cuidado.

La **activación sigmoideal** se utiliza en problemas de **clasificación binaria**. Su forma hace que la activación fluctúe entre las **dos clases** (activación entre 0 y 1). Cuanto mayor sea la **confianza** de la red en su predicción, esta estará más cercana a 0 o 1.

$$S(z) = \frac{1}{1 + e^{-z}} \quad (1)$$



Clasificación binaria: Binary crossentropy

$$Loss = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})) \quad (2)$$

$$Loss = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \quad (3)$$

*Cuando se trabaja con varias salidas al mismo tiempo (entrenamiento con *mini-batches*) se realiza un promediado de la pérdida.

Notebook de ejemplo, clasificación binaria

El siguiente notebook contiene un ejemplo de clasificador Convolutional Neural Network (CNN) para clasificación de imágenes rayos X para diagnostico de Covid.

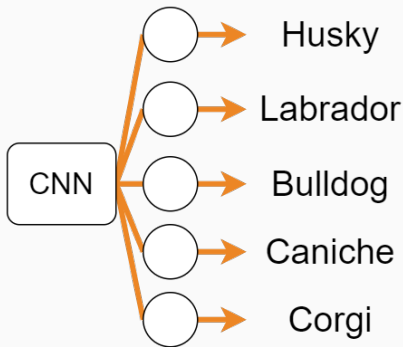


· 03.04-BinaryClass.ipynb

Clasificación multi-clase

Uno de los problemas más básicos que tratan las redes neuronales es la clasificación. Dentro de ella encontramos distintos tipos de problemas:

- Clasificación multi-clase

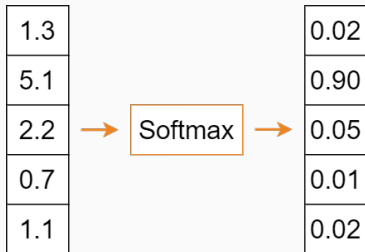


Clasificación multi-clase: activación de capa de salida

Dentro de los problemas de **clasificación** la salida de la red debe ser escogida con especialmente cuidado.

La **activación softmax** se utiliza en problemas de **clasificación multi-clase**. Su funcionamiento calcula la distribución de probabilidad de **n elementos**, de tal manera que la **suma** de sus probabilidades sea 1.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad (4)$$



Clasificación multi-clase con codificación one-hot: Categorical crossentropy

$$Loss = -y \log(\hat{y}) \quad (5)$$

$$Loss = - \sum_{i=1}^{\text{output size}} y_i \log(\hat{y}_i) \quad (6)$$

Cuando se trabaja con varias salidas al mismo tiempo (entreno con **mini-batches) se realiza un promediado de la pérdida.*

Clasificación multi-clase con codificación numérica: Sparse categorical crossentropy

$$Loss = -y \log(\hat{y}) \quad (7)$$

$$Loss = - \sum_{i=1}^{\text{output size}} y_i \log(\hat{y}_i) \quad (8)$$

Cuando se trabaja con varias salidas al mismo tiempo (entreno con **mini-batches) se realiza un promediado de la pérdida.*

Notebook de ejemplo, clasificación multi-clase

El siguiente notebook contiene un ejemplo de clasificador CNN para clasificación de imágenes de animales.

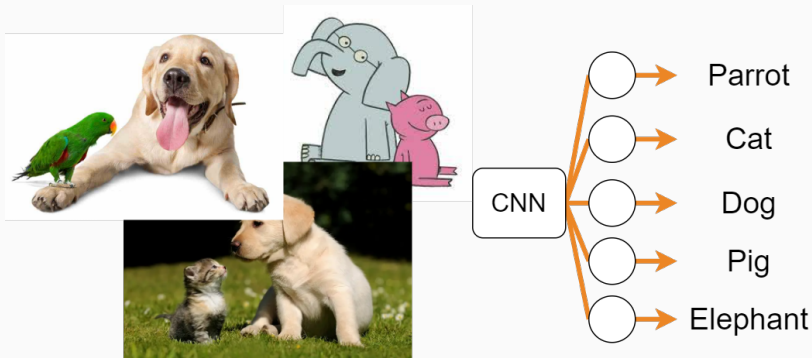


· 03.05-MultiClass.ipynb

Clasificación multi-etiqueta

Uno de los problemas más **básicos** que tratan las redes neuronales es la clasificación. Dentro de ella encontramos **distintos tipos de problemas**:

- **Clasificación multi-etiqueta**



Para implementar una clasificación multi-etiqueta con CNN se sigue los mismos pasos que para una binaria, **lo único que cambian son las etiquetas** de cada muestra (de cada imagen en este caso).

Por lo tanto, utilizamos **activación sigmoid** como función de activación de la capa de salida, y **binary crossentropy** como función de pérdida.

Notebook de ejemplo, clasificación multi-etiqueta

El siguiente notebook contiene un ejemplo de clasificador CNN para clasificación de imágenes de frutas.



· 03.06-MultiLabel.ipynb

Clasificación jerárquica

Uno de los problemas más **básicos** que tratan las redes neuronales es la clasificación. Dentro de ella encontramos **distintos tipos de problemas**:

- Clasificación jerárquica

