valgrAI

Artificial Intelligence for ICT professionals

Tensorflow y Keras

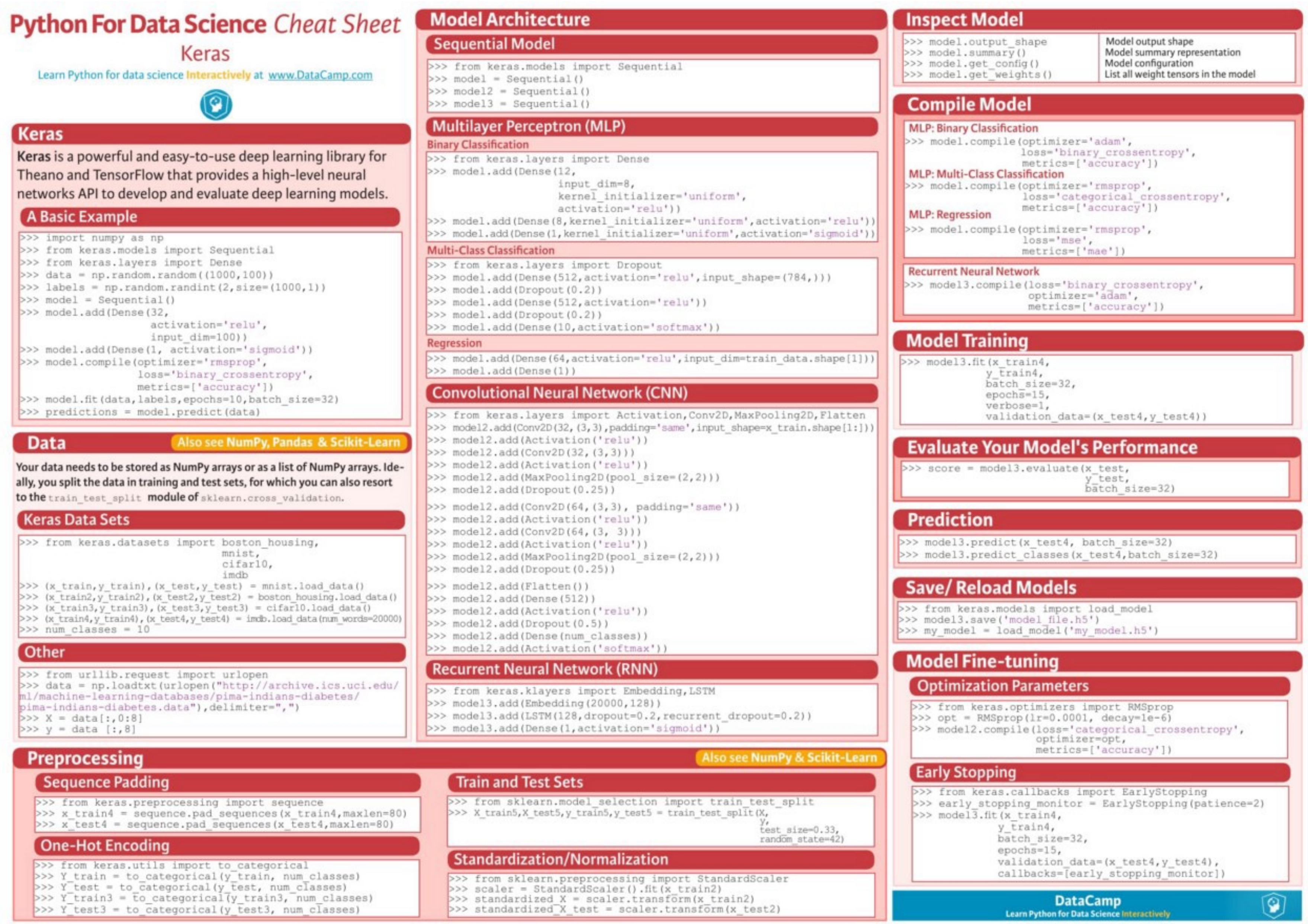
valgrai.eu

valgrai.eu

Tensorflow y Keras

- TensorFlow es una librería de **código abierto** para el aprendizaje automático profundo. Se puede utilizar en una amplia gama de tareas, pero se centra especialmente en el entrenamiento y la inferencia de redes neuronales profundas.
- TensorFlow fue desarrollado por el equipo de Google Brain para uso interno de Google. Fue lanzado bajo la licencia Apache 2.0 en 2015.
- TensorFlow 2.0 introdujo muchos cambios, siendo el más significativo TensorFlow eager, que cambió el esquema de diferenciación automática del gráfico computacional estático
- Alternativa principal: **pytorch** (Facebook)
- **Keras** fue creado por François Chollet como una capa a otros frameworks de bajo nivel con el fin de ser más fácil de usar, modular y extensible
- A partir de la versión 2.4 solo se usa con Tensorflow y ahora **forma parte de él**

Introducción



Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

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

MLP: Multi-Class Classification

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

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

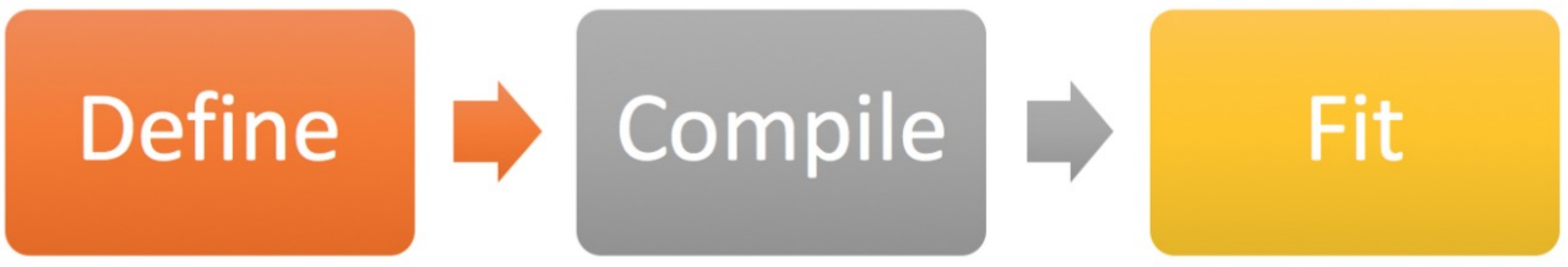
```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

DataCamp
Learn Python for Data Science Interactively

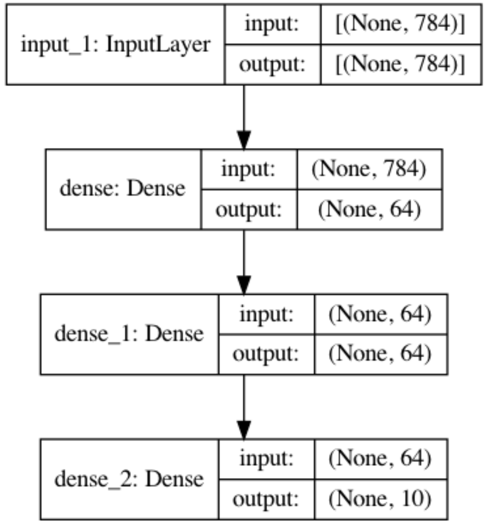
Introducción



Introducción

- En **Keras** se pueden crear modelos mediante el uso de la API **secuencial** (la más común) o mediante la API **funcional** (posibilidades más avanzadas)
- La **secuencial es la más común**, aunque hay casos en los que la funcional nos permite más posibilidades
- La funcional permite múltiples capas de entrada y salida, así como capas compartidas, lo que le permite construir estructuras de red realmente **complejas**

Sequential



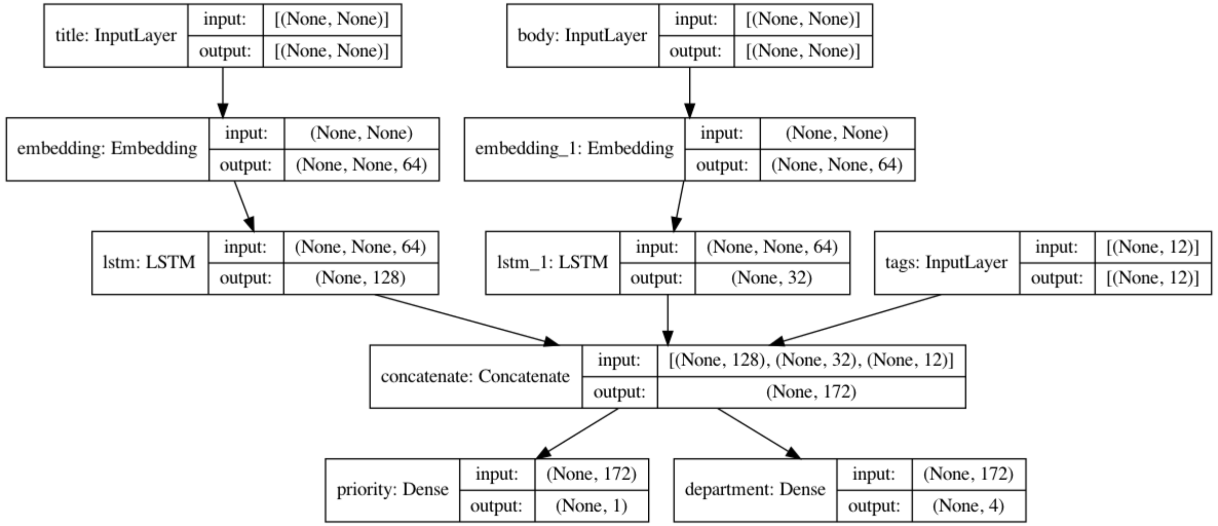
input_1: InputLayer
input: [(None, 784)]
output: [(None, 784)]

dense: Dense
input: (None, 784)
output: (None, 64)

dense_1: Dense
input: (None, 64)
output: (None, 64)

dense_2: Dense
input: (None, 64)
output: (None, 10)

Functional



title: InputLayer
input: [(None, None)]
output: [(None, None)]

body: InputLayer
input: [(None, None)]
output: [(None, None)]

embedding: Embedding
input: (None, None)
output: (None, None, 64)

embedding_1: Embedding
input: (None, None)
output: (None, None, 64)

lstm: LSTM
input: (None, None, 64)
output: (None, 128)

lstm_1: LSTM
input: (None, None, 64)
output: (None, 32)

tags: InputLayer
input: [(None, 12)]
output: [(None, 12)]

concatenate: Concatenate
input: [(None, 128), (None, 32), (None, 12)]
output: (None, 172)

priority: Dense
input: (None, 172)
output: (None, 1)

department: Dense
input: (None, 172)
output: (None, 4)

https://keras.io/guides/functional_api/

Sequential

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu',
input_shape=x_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(10, activation='softmax'))
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

model = Sequential([
    Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=x_train.shape[1:]),
    Conv2D(filters=32, kernel_size=(5,5), activation='relu'),
    MaxPool2D(pool_size=(2, 2)),
    Dropout(rate=0.25),
    Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    MaxPool2D(pool_size=(2, 2)),
    Dropout(rate=0.25),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(rate=0.5),
    Dense(10, activation='softmax')
])
```

Functional

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout, Input

inputs = Input(shape=x_train.shape[1:])

x = Conv2D(filters=32, kernel_size=(5,5), activation='relu')(inputs)
x = Conv2D(filters=32, kernel_size=(5,5), activation='relu')(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.25)(x)

x = Conv2D(filters=64, kernel_size=(3,3), activation='relu')(x)
x = Conv2D(filters=64, kernel_size=(3,3), activation='relu')(x)
x = MaxPool2D(pool_size=(2, 2))(x)
x = Dropout(rate=0.25)(x)

x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(rate=0.5)(x)
predictions = Dense(10, activation='softmax')(x)

model = Model(inputs=inputs, outputs=predictions)
```

Un ejemplo básico

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

data = np.random.random((1000,100))
labels = np.random.randint(2,size=(1000,1))

model = Sequential()

model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))

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

model.fit(data,labels,epochs=10,batch_size=32)

predictions = model.predict(data)
```

o....

```
import tensorflow as tf
```

y....

```
tk.keras.Sequential()
tf.keras.layers.Dense()
```

etc...

not real neurons, just input

o....

```
model.add(Input(shape(100,))
```

- 1 input layer (100)
- 1 hidden layer (32)
- 1 output layer (1)

mae, mse, custom metrics etc.

category_crossentropy, mse etc.

sgd, adam, RMSProp etc.

training process

Clasificación binaria

```
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


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

- 1 neurona única en la capa de salida 'sigmoide'
- Función de pérdida 'binary_crossentropy'
- valor de salida de 0 to 1 (<> 0.5)

Clasificación multiclase

Permitir datos de entrada n-dimensionales

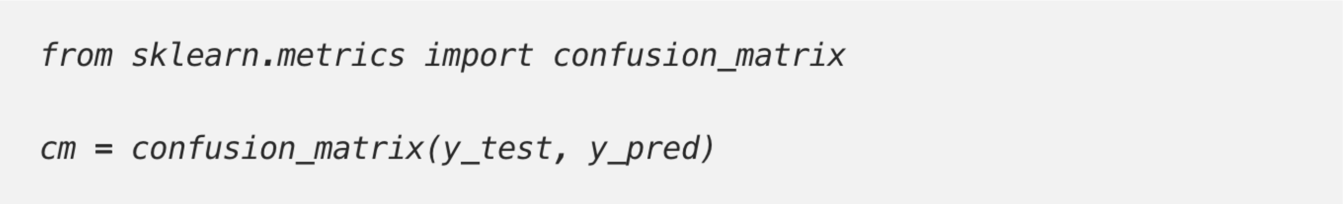
```
model.add(Dense(512,activation='relu',input_shape=(784,)))
model.add(Dropout(0.2)) #optional -> regularization
model.add(Dense(512,activation='relu'))
model.add(Dropout(0.2)) #optinal -> regularization
model.add(Dense(10,activation='softmax'))

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

- Tantas neuronas como categorías, activación 'softmax'
- Función de pérdida 'categorizal_crossentropy'
- valor de salida de 0 to 1 (<> 0.5)
- Toma el valor máximo de todos ellos

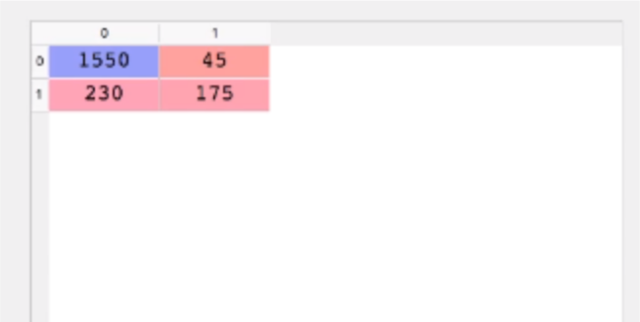
Clasificación multiclase

```
y_pred = classifier.predict(X_test)

y_pred = (y_pred > 0.5)
```

Métricas de clasificación



```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
```



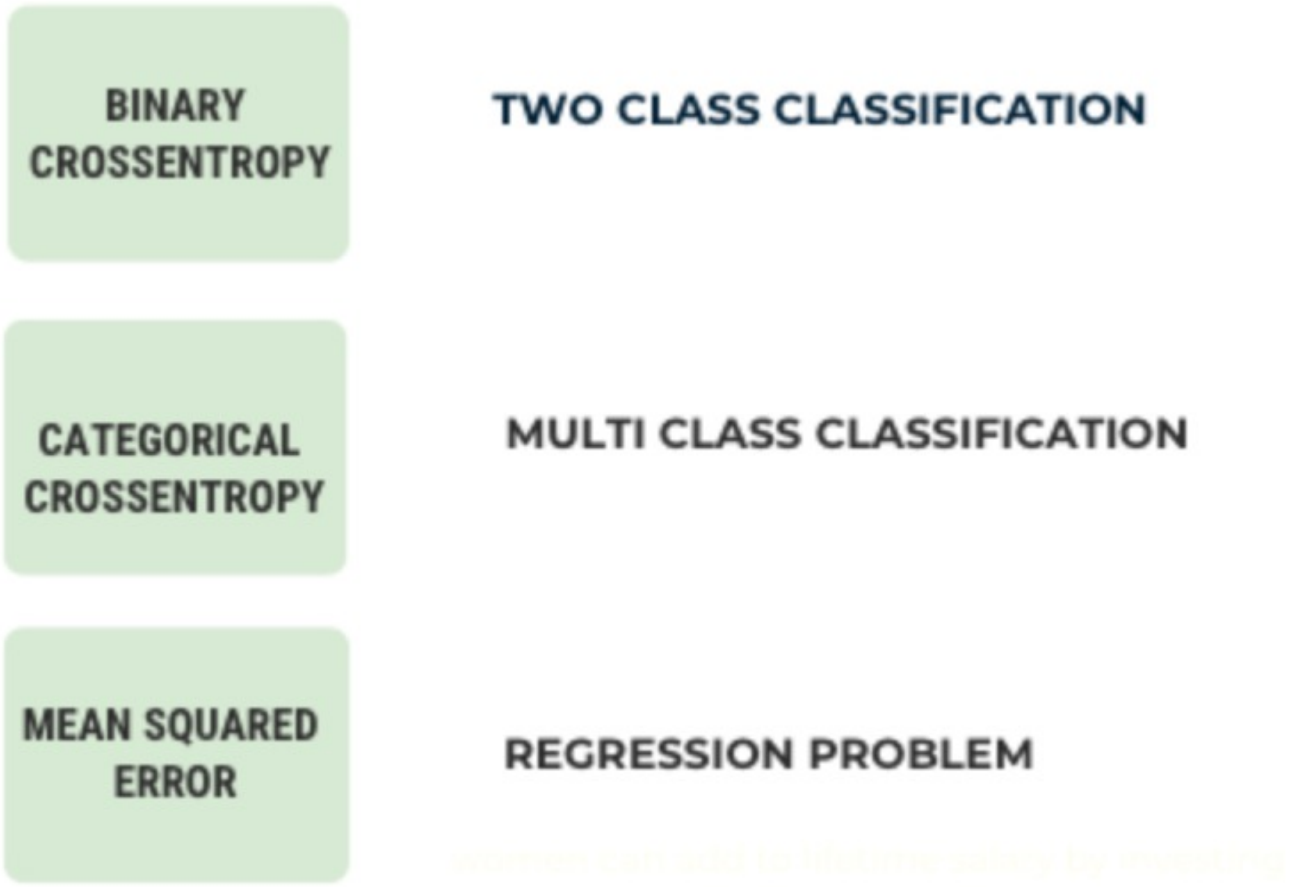
	0	1
0	1550	45
1	230	175

Regresión

```
model.add(Dense(64, activation='relu', input_dim=10)
model.add(Dense(1))  # activation -> Linear by default -> perfect for regression

model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

- Tantas neuronas como categorías, activación 'softmax'
- Función de pérdida 'MSE' (error cuadrático medio)
- valor de salida: el 'valor' predicho

Summary



BINARY CROSSENTROPY

TWO CLASS CLASSIFICATION

CATEGORICAL CROSSENTROPY

MULTI CLASS CLASSIFICATION

MEAN SQUARED ERROR

REGRESSION PROBLEM

women can add to lifetime salary by investing

Información sobre el modelo

```
model.output_shape
```

`model.summary()`

```
model.get_weights()
```

etc.

Entrenamiento de modelos

```
model.fit(X, y, batch_size=32, epochs=15, verbose=2, validation_data(X_test, y_test))
```

O …

`validation_split=0.2`

si usamos solo el conjunto de datos de entrenamiento

Evaluación del rendimiento del modelo

```
score = model.evaluate(X_test, Y_test, batch_size=32)
```

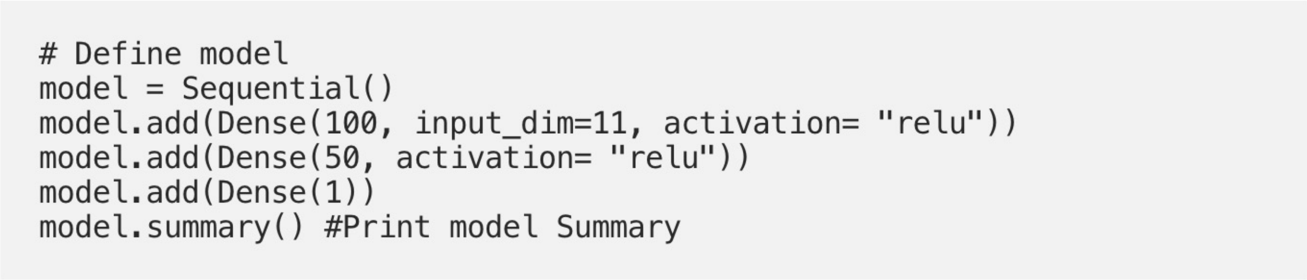
Prediction, fine-tuning, early stopping....

```
# prediction
y_pred = model.predict(X_test)
```

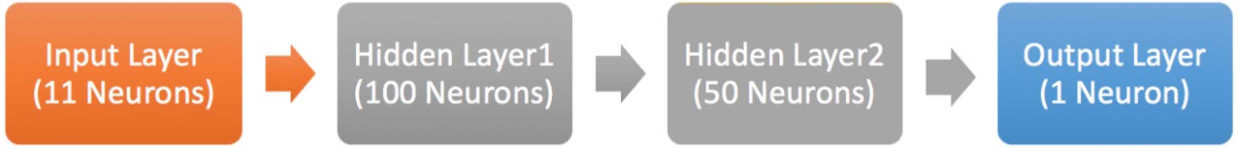
```
# optimizers, regularization and more....
from tensorflow.keras.optimizers import RMSprop
opt = RMSprop(lr=0.0001, decay=1e-6)

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

```
# Early stopping
from tensorflow.keras.callbacks import EarlyStopping
early_stopping_monitor = EarlyStopping(patience=3)

model.fit(x_train, y_train, batch_size=32, epochs=15, validation_split=0.2,
callbacks=[early_stopping_monitor])
```

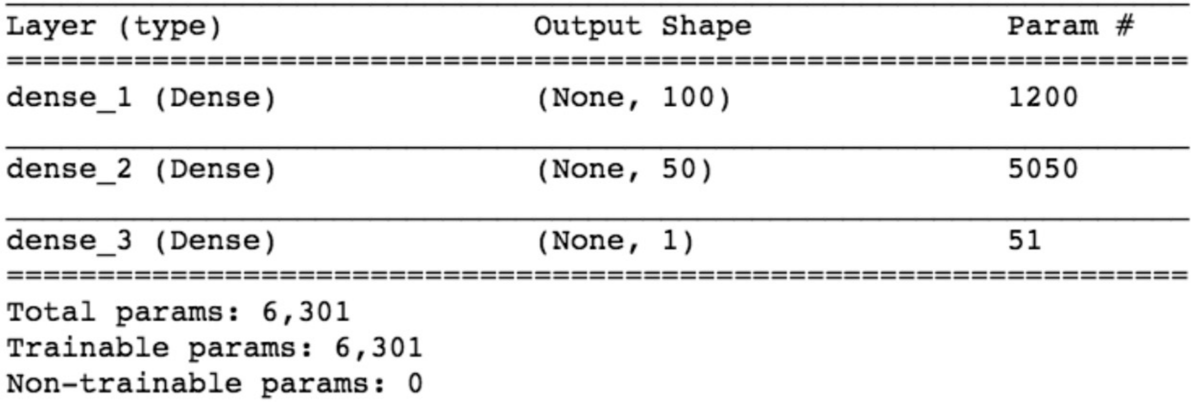
Ejemplos



```
# Define model
model = Sequential()
model.add(Dense(100, input_dim=11, activation= "relu"))
model.add(Dense(50, activation= "relu"))
model.add(Dense(1))
model.summary() #Print model Summary
```

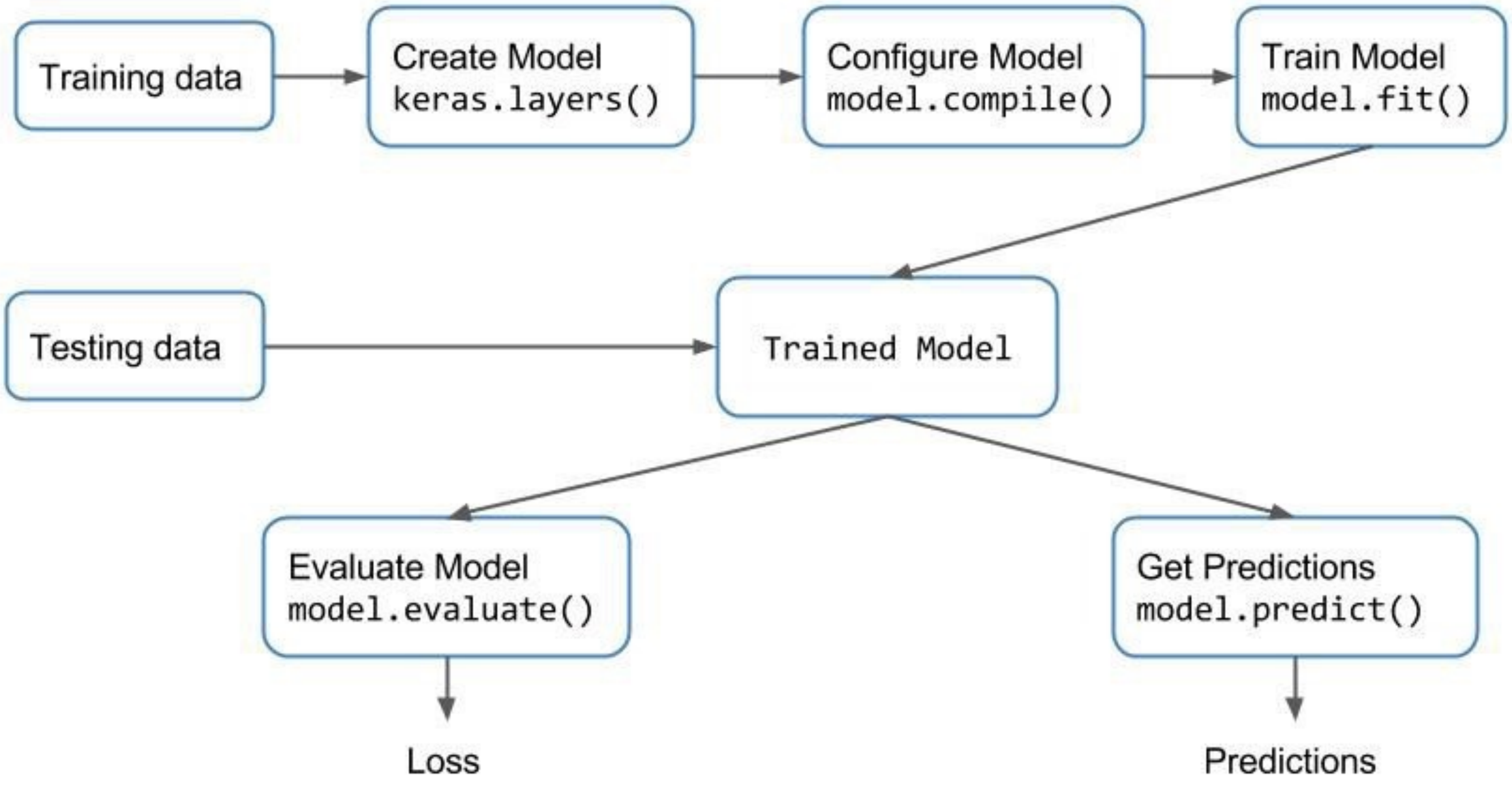


Input Layer (11 Neurons) → Hidden Layer1 (100 Neurons) → Hidden Layer2 (50 Neurons) → Output Layer (1 Neuron)



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 100)               1200
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 51
=================================================================
Total params: 6,301
Trainable params: 6,301
Non-trainable params: 0
```

Keras workflow



Valencian Graduate School and Research Network of Artificial Intelligence

valgrAI

valgrai.eu

valgrai.eu

Valencian Graduate School and Research Network of Artificial Intelligence