

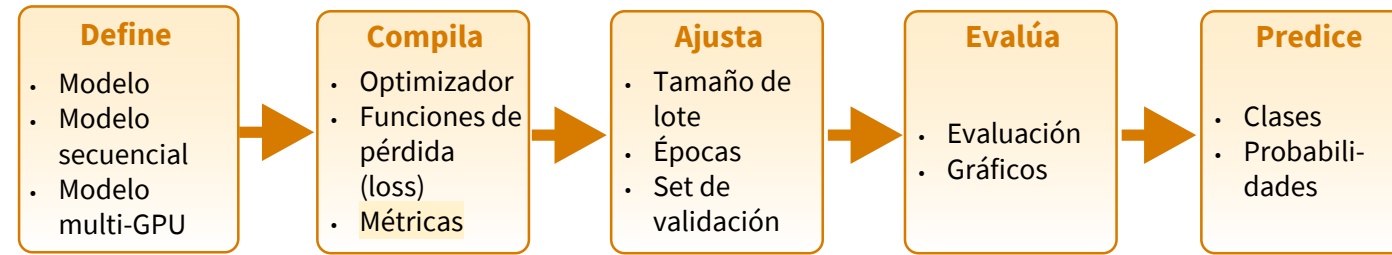
Aprendizaje Profundo con Keras :: GUÍA RÁPIDA



Introducción

[Keras](#) es una API de redes neuronales profundas desarrollada con el objetivo de facilitar la experimentación rápida. Soporta múltiples plataformas, incluyendo TensorFlow, CNTK y Theano.

TensorFlow es una librería matemática de bajo nivel para la construcción de arquitecturas de redes neuronales profundas. El paquete de R `keras` facilita el uso de Keras y Tensorflow en R.



<https://keras.rstudio.com>

<https://www.manning.com/books/deep-learning-with-r>

El “¡Hola, mundo!” del aprendizaje profundo

Trabajando con modelos en keras

DEFINE UN MODELO

`keras_model()` crea un modelo de Keras

`keras_model_sequential()` crea un modelo de Keras compuesto por un apilado lineal de capas

`multi_gpu_model()` replica un modelo en GPUs distintas.

COMPILA UN MODELO

`compile(object, optimizer, loss, metrics = NULL)` configura un modelo de Keras para entrenamiento

AJUSTA UN MODELO

`fit(object, x = NULL, y = NULL, batch_size = NULL, epochs = 10, verbose = 1, callbacks = NULL, ...)` entrena un modelo de Keras para un número fijo de épocas (iteraciones)

`fit_generator()` ajusta el modelo con datos producidos en lotes por un generador

`train_on_batch()` `test_on_batch()` actualización del gradiente o evaluación del modelo sobre un único lote de muestras

EVALÚA UN MODELO

`evaluate(object, x = NULL, y = NULL, batch_size = NULL)` evalúa un modelo de Keras

`evaluate_generator()` evalúa el modelo sobre datos generados

PREDICE

`predict()` se usa para generar predicciones a partir de un modelo de Keras

`predict_proba()` y `predict_classes()` sirven para generar predicciones de probabilidades o de clases

`predict_on_batch()` devuelve predicciones para un lote único de muestras

`predict_generator()` genera predicciones para muestras de entrada de un generador de datos

OTRAS OPERACIONES

`summary()` imprime el resumen de un modelo de Keras

`export_savedmodel()` exporta un modelo guardado

`get_layer()` recupera una capa a partir de su nombre (único) o su índice

`pop_layer()` remueve la última capa de un modelo

`save_model_hdf5()` y `load_model_hdf5()` guarda/carga modelos usando archivos HDF5

`serialize_model()` devuelve un objeto R “crudo” conteniendo una versión HDF5 del modelo Keras

`unserialize_model()` devuelve un modelo de Keras

`clone_model()` clona una instancia de modelo

`freeze_weights()`; `unfreeze_weights()` congela/descongela los pesos

CAPAS

`layer_input()` la capa de entrada

`layer_dense()` agrega una capa NN densamente conectada a la salida.

`layer_activation()` aplica una función de activación a una salida.

`layer_dropout()` aplica Dropout a la capa de entrada.

`layer_reshape()` ajusta la salida a un cierto formato

`layer_permute()` permuta las dimensiones de una entrada de acuerdo a una patrón determinado

`layer_repeat_vector()` repite la entrada n veces

`layer_lambda(object, f)` envuelve una expresión arbitraria como una capa

`layer_activity_regularization()` aplica una actualización de la función de costo (l1 o l2) basado en la actividad de la entrada

`layer_masking()` permite saltar un paso de tiempo en todas las capas si la entrada coincide con valores de máscara definidos

`layer_flatten()` aplanar una entrada

INSTALACIÓN

El paquete de R `keras` usa la librería Keras de Python. Todos los prerequisites se pueden instalar directamente desde R.

https://keras.rstudio.com/reference/install_keras.html

```
library(keras)
install_keras()
```

Usa `?install_keras` para las instrucciones con GPU

Esto instala las librerías necesarias en un entorno Anaconda o en un entorno virtual 'r-tensorflow'.

ENTRENA UN SISTEMA DE RECONOCIMIENTO DE IMÁGENES SOBRE LOS DATOS MNIST

```
# input layer: usa imágenes MNIST
mnist <- dataset_mnist()
x_train <- mnist$train$x; y_train <- mnist$train$y
x_test <- mnist$test$x; y_test <- mnist$test$y
```

re-organización y re-escalado

```
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
x_train <- x_train / 255; x_test <- x_test / 255
```

```
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

definir el modelo y las capas

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu',
    input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')
```

compilar (definir funciones de pérdida y optimizadores)


```
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
```


entrenar (ajustar)


```
model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)
model %>% evaluate(x_test, y_test)
model %>% predict_classes(x_test)
```


Más capas


CONVOLUCIÓN

**layer_conv_1d()** 1D, ej. convolución temporal

**layer_conv_2d()** 2D, ej. convolución espacial sobre imágenes

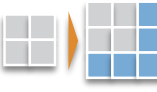
**layer_conv_2d_transpose()** transpuesta de la convolución 2D (deconvolución)


**layer_conv_3d_transpose()** transpuesta de la convolución 3D (deconvolución)


**layer_conv_3d()** 3D, ej. convolución espacial sobre volúmenes

layer_conv_lstm_2d() LSTM convolucional

layer_separable_conv_2d() convolución 2D separable


**layer_upsampling_1d()**
layer_upsampling_2d()
layer_upsampling_3d() repite datos (upsampling) según sus dimensiones

**layer_zero_padding_1d()**
layer_zero_padding_2d()
layer_zero_padding_3d() Agrega ceros al inicio/fin/bordes de los datos de la capa de entrada

**layer_cropping_1d()**
layer_cropping_2d()
layer_cropping_3d() Recorta los datos de la capa de entrada

AGREGACIÓN (POOLING)


**layer_max_pooling_1d()**
layer_max_pooling_2d()
layer_max_pooling_3d() Agrega en ventanas (1D a 3D) usando el máximo (max pooling)


**layer_average_pooling_1d()**
layer_average_pooling_2d()
layer_average_pooling_3d() Agrega en ventanas usando el promedio


**layer_global_max_pooling_1d()**
layer_global_max_pooling_2d()
layer_global_max_pooling_3d() Agrega usando el máximo global


**layer_global_average_pooling_1d()**
layer_global_average_pooling_2d()
layer_global_average_pooling_3d() Agrega usando el promedio global


ACTIVACIÓN

**layer_activation(object, activation)**
Aplica una función de activación a una salida


**layer_activation_leaky_relu()**
Aplica una ReLU con un pequeño gradiente


**layer_activation_parametric_relu()**
Aplica una ReLU paramétrica

**layer_activation_thresholded_relu()**
Aplica una ReLU activada según umbrales

**layer_activation_elu()**
Aplica una ReLU exponencial

ABANDONO (DROPOUT)

**layer_dropout()**
Aplica *dropout* a la entrada

**layer_spatial_dropout_1d()**
layer_spatial_dropout_2d()
layer_spatial_dropout_3d() Versiones para *dropout* espacial 1D a 3D

CAPAS RECURRENTE

**layer_simple_rnn()**
Implementa una arquitectura RNN completamente conectada donde la salida retro-alimenta la entrada

layer_gru()
Implementa una arquitectura GRU (unidad cerrada recurrente) de Cho et al 2014

layer_cudnn_gru()
implementación GRU rápida basada en CuDNN

layer_lstm()
Implementa una arquitectura Long-Short Term Memory de Hochreiter 1997

layer_cudnn_lstm()
Implementación LSTM rápida basada en CuDNN

CAPAS CONECTADAS LOCALMENTE

layer_locally_connected_1d()
layer_locally_connected_2d()
Son similares a las convolutivas pero los pesos no se comparten, son filtros diferentes por cada uno de los parches

Preprocesamiento

PREPROCESAMIENTO DE SECUENCIAS

pad_sequences()
Completa cada secuencia a la misma longitud (longitud de la secuencia más larga)

skipgrams()
Genera pares de palabras usando skip-gram

make_sampling_table()
Genera una tabla de muestreo probabilística basada en rankings de palabras

PREPROCESAMIENTO DE TEXTOS

text_tokenizer() función para separar (tokenizar) textos

fit_text_tokenizer() actualiza el vocabulario interno de tokens

save_text_tokenizer(); load_text_tokenizer() guarda un tokenizador a un archivo externo ; carga un tokenizador desde un archivo externo

texts_to_sequences(); texts_to_sequences_generator() transforma cada texto en una secuencia de enteros

texts_to_matrix(); sequences_to_matrix() convierte una lista de secuencia a una matriz

text_one_hot() codifica un texto con one-hot usando índices de palabras

text_hashing_trick() convierte un texto a una secuencia de índices en un espacio hash de tamaño fijo }

text_to_word_sequence() convierte texto a una secuencia de palabras o tokens

PREPROCESAMIENTO DE IMAGEN

image_load() carga una imagen al formato PIL.

flow_images_from_data()
flow_images_from_directory() Genera lotes de datos aumentados/normalizados de imágenes y etiquetas de manera individual o desde un directorio

image_data_generator() Genera mini-lotes de datos de imagen con aumento en tiempo real.

fit_image_data_generator() ajusta los estadísticos internos del generador de datos de imagen a partir de una muestra dada

generator_next() Recupera el siguiente ítem

image_to_array(); image_array_resize()
image_array_save() convierte imágenes multidimensionales en arreglos 3D; cambia el tamaño del arreglo; guarda el arreglo en un archivo externo



Modelos Pre-entrenados

Las aplicaciones de Keras son modelos de aprendizaje profundo que están disponibles con sus pesos. Estos modelos pueden ser usados para hacer predicciones, extracción de características y ajustes finos.

application_xception()
xception_preprocess_input()
Modelo Xception v1

application_inception_v3()
inception_v3_preprocess_input()
Modelo Inception v3, con pesos pre-entrenados sobre ImageNet

application_inception_resnet_v2()
inception_resnet_v2_preprocess_input()
Modelo Inception-ResNet v2, con pesos pre-entrenados sobre ImageNet

application_vgg16(); application_vgg19()
Modelos VGG16 y VGG19

application_resnet50()
Modelo ResNet50

application_mobilenet()
mobilenet_preprocess_input()
mobilenet_decode_predictions()
mobilenet_load_model_hdf5()
Modelo MobileNet

IMAGENET

[ImageNet](#) es una enorme base de datos de imágenes etiquetadas, muy usada para aprendizaje profundo

magenet_preprocess_input()
imagenet_decode_predictions()
Preprocesa un tensor de imágenes de ImageNet, y decodifica las predicciones

Retrollamadas (Callbacks)

Una retro-llamada es un conjunto de funciones que se pueden aplicar en determinados estadios del proceso de entrenamiento. Se pueden usar durante el entrenamiento para tener una vista de los estados internos y de las estadísticas del modelo

callback_early_stopping() Detiene el entrenamiento cuando el parámetro monitoreado ha dejado de mejorar

callback_learning_rate_scheduler() Muestra la tasa de aprendizaje en cada época

callback_tensorboard() Visualizaciones básicas de TensorBoard

