

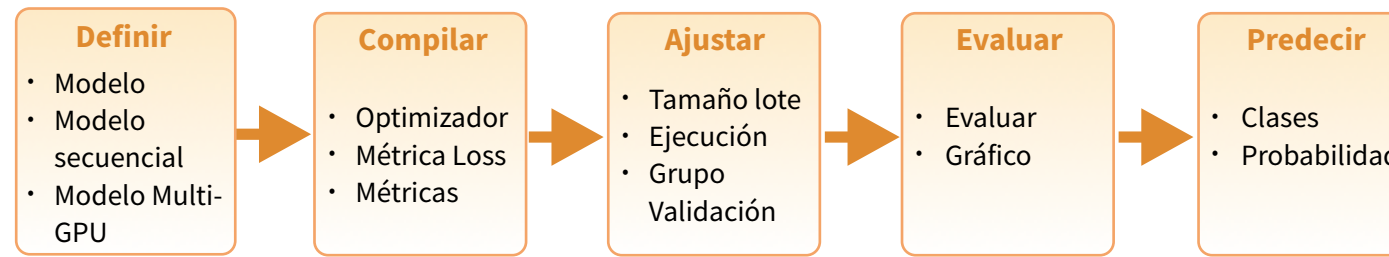
Aprendizaje Profundo con Keras :: GUÍA RÁPIDA



Introducción

Keras es un API de red neuronal de alto nivel desarrollada con el foco de facilitar una rápida experimentación. Soporta múltiples plataformas, incluyendo TensorFlow, CNTK y Theano.

TensorFlow es una librería matemática de bajo nivel para la construcción de arquitecturas de aprendizaje profundo. El paquete de R **keras** permite el uso fácil 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

INSTALACIÓN

El paquete de R **keras** usa la librería **keras** de Python. Se pueden instalar todos los pre-requisitos desde R

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

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

Mira ?keras_install para las instrucciones sobre GPU

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

Trabajando con modelos en keras

DEFINIR UN MODELO

keras_model() Modelo Keras

keras_model_sequential() Model Keras compuesto de un conjunto apilado de capas lineales

multi_gpu_model() Replica un modelo en diferentes GPUs

COMPILA UN MODELO

compile(object, optimizer, loss, metrics = NULL) Configura un modelo Keras para entrenamiento

AJUSTA UN MODELO

fit(object, x = NULL, y = NULL, batch_size = NULL, epochs = 10, verbose = 1, callbacks = NULL, ...) Entrena un modelo Keras durante un número de epochs (iteraciones)

fit_generator() Ajusta el modelo sobre datos producidos en lotes por un generador

train_on_batch() **test_on_batch()** Actualización sobre una sola iteración del gradiente o evaluación del modelo sobre un lote de muestras

EVALUAR UN MODELO

evaluate(object, x = NULL, y = NULL, batch_size = NULL) Evalúa un modelo Keras

evaluate_generator() Evalúa el modelo sobre datos generados

PREDECIR

predict() Genera predicciones de un modelo Keras

predict_proba() and **predict_classes()** Genera predicciones en forma de probabilidades o clases para las muestras de entrada

predict_on_batch() Devuelve predicciones para un lote de muestras

predict_generator() Genera predicciones para las muestras de entrada de un generador de datos

OTRAS OPERACIONES SOBRE LOS MODELOS

summary() Resumen de un modelo Keras

export_savedmodel() Exporta un modelo guardado

get_layer() Recupera una capa basándose en su nombre (único) o un índice

pop_layer() Elimina la última capa de un modelo

save_model_hdf5(); load_model_hdf5() Salva/Carga modelos usando archivos HDF5

serialize_model(); unserialize_model() Serializa un modelo a un objeto R

clone_model() Clona una instancia del modelo

freeze_weights(); unfreeze_weights() Congela y descongela los pesos

CAPAS BASE

layer_input() Capa de entrada

layer_dense() Añade una capa NN densamente-conectada a una salida

layer_activation() Aplica una función de activación a una salida

layer_dropout() Aplica Dropout a la entrada

layer_reshape() Reajusta una salida a una determinada forma

layer_permute() Permuta las dimensiones de una entrada de acuerdo a un patrón determinado

layer_repeat_vector() Repite la entrada n veces


layer_lambda(object, f) Envuelve una expresión arbitraria como una capa

layer_activity_regularization() Capa que aplica una actualización a la función de costes basada en la actividad de la entrada

layer_masking() Enmascara una secuencia usando un valor de máscara para saltar saltos de tiempo

layer_flatten() Aplana una entrada

ENTRENA UN SISTEMA DE RECONOCIMIENTO DE IMÁGENES SOBRE LOS DATOS MNIST

```
#Input capa: usar 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
```

Ajuste 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 pérdida y optimizar)







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

Entrenamiento (Ajuste)





```
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 tipos de capas






CAPAS CONVOLUCIONADAS

-  **layer_conv_1d()** 1D, e.g. convolución temporal
-  **layer_conv_2d_transpose()** Transpuesta 2D (deconvolución)
-  **layer_conv_2d()** 2D, e.g. convolución espacial sobre imágenes
-  **layer_conv_3d_transpose()** Transpuesta 3D (deconvolution)
-  **layer_conv_3d()** 3D, e.g. convolución espacial sobre volúmenes
-  **layer_conv_lstm_2d()** Convolución LSTM
-  **layer_separable_conv_2d()** Separable en profundidad 2D
-  **layer_upsampling_1d()**
layer_upsampling_2d()
layer_upsampling_3d() Capa sobremuestreo
-  **layer_zero_padding_1d()**
layer_zero_padding_2d()
layer_zero_padding_3d() Capa zero-relleno
-  **layer_cropping_1d()**
layer_cropping_2d()
layer_cropping_3d() Capa de corte



CAPAS AGRUPACIÓN

-  **layer_max_pooling_1d()**
layer_max_pooling_2d()
layer_max_pooling_3d() Agrupación máxima de 1D a 3D
-  **layer_average_pooling_1d()**
layer_average_pooling_2d()
layer_average_pooling_3d() Agrupación media de 1D a 3D
-  **layer_global_max_pooling_1d()**
layer_global_max_pooling_2d()
layer_global_max_pooling_3d() Agrupación máxima global
-  **layer_global_average_pooling_1d()**
layer_global_average_pooling_2d()
layer_global_average_pooling_3d() Agrupación media global






CAPAS DE ACTIVACIÓN

-  **layer_activation(object, activation)** Aplica una función de activación a una salida
-  **layer_activation_leaky_relu()** Versión leaky de una unidad lineal rectificadora
-  **layer_activation_parametric_relu()** Unidad lineal rectificadora paramétrica
-  **layer_activation_thresholded_relu()** Unidad lineal rectificadora con umbral
-  **layer_activation_elu()** Unidad lineal exponencial


CAPAS DE ABANDONO

-  **layer_dropout()** Aplica un abandono a una entrada
-  **layer_spatial_dropout_1d()**
layer_spatial_dropout_2d()
layer_spatial_dropout_3d() Versión espacial 1D a 3D de un abandono

CAPAS RECURRENTE

-  **layer_simple_rnn()** RNN completa-conectada donde la salida retro-alimenta la entrada
-  **layer_gru()** Unidad recurrente tipo puerta - Cho et al
-  **layer_cudnn_gru()** Implementación rápida de unidad GRU basada en CuDNN
-  **layer_lstm()** Unidad memoria largo-corto plazo - Hochreiter 1997
-  **layer_cudnn_lstm()** Implementación rápida LSTM basada en CuDNN

CAPAS CONECTADAS LOCALMENTE

-  **layer_locally_connected_1d()**
layer_locally_connected_2d() Similares a convolución, pero los pesos no son compartidos, i.e. filtros diferentes para cada parche

Preprocesado

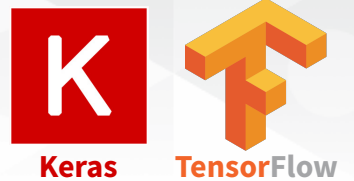
PREPROCESADO DE SECUENCIA

- pad_sequences()** Completa cada secuencia a la misma longitud (longitud de la secuencia más larga)
- skipgrams()** Genera skipgram pares de palabras
- make_sampling_table()** Genera una tabla de palabras basadas en un ranking de probabilidades
- PREPROCESADO DE TEXTO**
- text_tokenizer()** Utilidad de separación de texto
- fit_text_tokenizer()** Actualiza internamente el vocabulario de tokens
- save_text_tokenizer(); load_text_tokenizer()** Guarda un texto separado en un fichero externo
- texts_to_sequences(); texts_to_sequences_generator()** Transforma cada texto en textos a secuencias de enteros
- texts_to_matrix(); sequences_to_matrix()** Convierte una secuencia de listas en una matriz

- text_one_hot()** Codifica one-hot un texto a 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 un texto a una secuencia de palabras (tokens)

PREPROCESADO DE IMÁGENES

- image_load()** Carga una imagen en formato PIL
- flow_images_from_data()**
flow_images_from_directory() Genera lotes de datos aumentados/normalizados de imágenes, etiquetas o de un directorio
- image_data_generator()** Genera minilotes de datos de imágenes con aumento en tiempo real.
- fit_image_data_generator()** Ajusta las estadísticas del generador interno de datos a partir de una muestra de datos
- generator_next()** Recupera el siguiente item
- image_to_array(); image_array_resize()**
image_array_save() Representación 3D de un array



Modelos pre-entrenados

Las aplicaciones Keras son modelos de aprendizaje profundo que están disponibles con sus pesos. Estos modelos se pueden usar para predicción, extracción de rasgos, y ajuste fino.

- application_xception()**
xception_preprocess_input() Modelo Xception v1
- application_inception_v3()**
inception_v3_preprocess_input() Modelo Inception v3, con pesos pre-entrenados en ImageNet
- application_inception_resnet_v2()**
inception_resnet_v2_preprocess_input() Modelo Inception-ResNet v2, con pesos entrenados en ImageNet
- application_vgg16(); application_vgg19()** Modelos VGG16 and VGG19
- application_resnet50()** Modelo ResNet50
- application_mobilenet()**
mobilenet_preprocess_input()
mobilenet_decode_predictions()
mobilenet_load_model_hdf5() Arquitectura de modelo MobileNet

IMAGENET

[ImageNet](https://www.image-net.org/) es una gran base de datos de imágenes etiquetadas, usada de forma intensiva en aprendizaje profundo

- imagenet_preprocess_input()**
imagenet_decode_predictions() Preprocesa un tensor de imágenes de ImageNet, y decodifica predicciones

Retro-llamadas

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

- callback_early_stopping()** Para el entrenamiento cuando una cantidad monitorizada ha dejado de mejorar
- callback_learning_rate_scheduler()** Ratio de aprendizaje del planificador
- callback_tensorboard()** Visualizaciones básicas del TensorBoard

