TL02 Keras

Índice

1 Introducción: https://keras.io/getting_started

Keras 2: librería Python de aprendizaje profundo basada en TensorFlow

Keras 3: versión reciente con multi-backend (JAX, TensorFlow y PyTorch)

Creador y desarrollador principal: François Chollet https://fchollet.com

Documentación: https://keras.io

Github: https://github.com/keras-team

Prueba en Colab: https://keras.io/getting_started/intro_to_keras_for_engineers

Tal vez sea necesario actualizar tensorflow y keras en una celda inicial de código:

In [ ]:
```
!pip install tensorflow --upgrade --quiet
!pip install keras --upgrade --quiet
```

Backend benchmarks: mejores resultados con Keras 3 y JAX o TF https://keras.io/getting_started/benchmarks

Ecosistema Keras: https://keras.io/getting_started/ecosystem

- **KerasTuner:** librería para el ajuste de hiper-parámetros
- **KerasNLP:** librería de procesamiento de lenguaje natural
- **KerasCV:** librería de visión por ordenador
- **AutoKeras:** sistema de AutoML basado en Keras

2 Guías: https://keras.io/guides

Uso: cuadernos jupyter fácilmente ejecutables en Colab

- The Functional API
- The Sequential model
- Making new layers & models via subclassing
- Training & evaluation with the built-in methods
- etc.

KerasTuner:

- Getting started with KerasTuner
- Distributed hyperparameter tuning with KerasTuner
- etc.

KerasCV:

- Use KerasCV to assemble object detection pipelines
- Use KerasCV to train powerful image classifiers
- etc.

KerasNLP:

- Getting Started with KerasNLP
- Pretraining a Transformer from scratch with KerasNLP
- Uploading Models with KerasNLP

3 API: https://keras.io/api

Models API: Model class, Sequential class, Model training APIs, Saving & serialization

Layers API: Base Layer class, Layer activations, Layer weights, Core layers, Convolution layers, Pooling layers, etc.

Callbacks API: Base, ModelCheckpoint, EarlyStopping, LearningRateScheduler, ReduceLROnPlateau, etc.

Ops API: NumPy ops, NN ops, Linear algebra ops, Core ops, Image ops, FFT ops

Optimizers: SGD, RMSprop, Adam, AdamW, etc.

Metrics: Base class, Accuracy, Probabilistic, Regression, Image segmentation, etc.

Losses: Probabilistic losses, Regression losses, Hinge losses

Data loading: Image data, Timeseries data, Text data, Audio data

Built-in datasets: MNIST, CIFAR10, CIFAR100, IMBD, Reuters newswire, Fashion MNIST, California Housing price

Misc.: Mixed precision, Multi-device distribution, RNG API

Utilities: Model plotting, Structured data preprocessing, Tensor, Python & NumPy, Keras configuration

Keras Applications: EfficientNet, ResNet, DenseNet, InceptionV3, InceptionResNetV2, etc.

KerasTuner: HyperParameters, Tuners, Oracles, HyperModels, Errors

KerasCV: Layers, Models, Bounding box formats and utilities, Losses

KerasNLP: Pretrained Models, Models API, Tokenizers, Preprocessing Layers, Modeling Layers, Samplers, Metrics

4 Ejemplos: https://keras.io/examples

Recomendados: marcados con una estrella y V3 (Keras 3)

CV: clasificación de imágenes, segmentación de imágenes, subtitulado de imágenes, etc.

NLP: clasificación de texto, traducción automática, etc.

Datos estructurados: clasificación de datos estructurados, etc.

Series temporales: clasificación de series temporales, etc.

Aprendizaje profundo generativo: generación de imágenes, generación de texto, etc.

Audio: reconocimiento automático del habla, etc.

Otros: aprendizaje por refuerzo, grafos

Recetas rápidas: consejos de uso, etc.

5 Un MLP sencillo para MNIST

Inicialización: librerías y semilla para la generación de números aleatorios

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras
keras.utils.set_random_seed(23)
```

Lectura de MNIST:

In [ ]:
```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
print(f'Lectura: train {x_train.shape} {y_train.shape} test {x_test.shape} {y_test.shape}')

input_dim = x_train.shape[1] * x_train.shape[2]
x_train = x_train.reshape(-1, input_dim).astype("float32")
x_test = x_test.reshape(-1, input_dim).astype("float32")
print(f'Reformato 1: train {x_train.shape} {y_train.shape} test {x_test.shape} {y_test.shape}')

x_train_max = np.max(x_train)
x_train /= x_train_max
x_test /= x_train_max
print(f'Normalización [0,1]: max = {x_train_max}')

num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
print(f'Reformato 2: train {x_train.shape} {y_train.shape} test {x_test.shape} {y_test.shape}')
```

```
Lectura: train (60000, 28, 28) (60000,) test (10000, 28, 28) (10000,)
Reformato 1: train (60000, 784) (60000,) test (10000, 784) (10000,)
Normalización [0,1]: max = 255.0
Reformato 2: train (60000, 784) (60000, 10) test (10000, 784) (10000, 10)
```

Definición del modelo: MLP con una capa oculta de ReLUs y capa de salida softmax

In [ ]:
```
M = keras.Sequential([keras.Input(shape=(input_dim,)),
    keras.layers.Dense(800, activation='relu'),
    keras.layers.Dense(num_classes, activation='softmax')])
M.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 800)	628,000
dense_1 (Dense)	(None, 10)	8,010

Total params: 636,010 (2.43 MB)

Trainable params: 636,010 (2.43 MB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
print(f'Número de parámetros de la capa oculta: {784*800 + 800}')
print(f'Número de parámetros de la capa de salida: {800*10 + 10}')
```

```
Número de parámetros de la capa oculta: 628000
Número de parámetros de la capa de salida: 8010
```

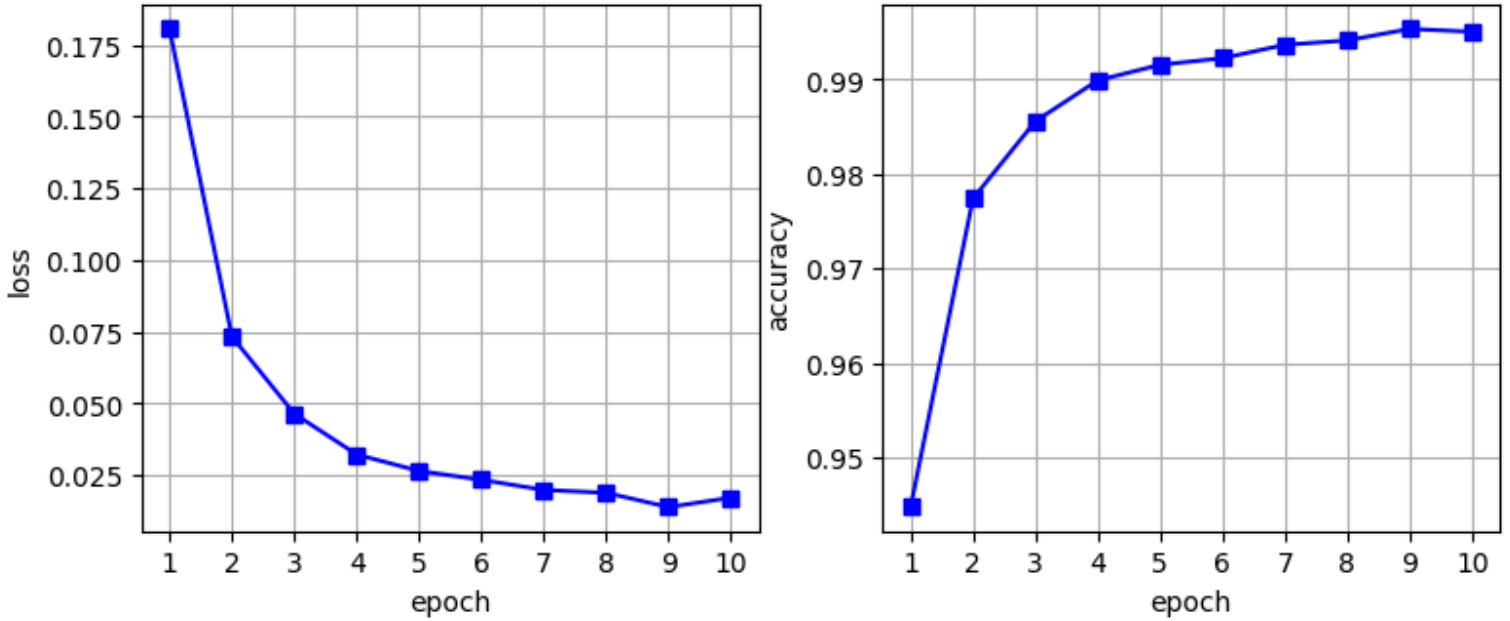
Compilación del modelo: definimos pérdida, optimizador y métrica

In [ ]:
```
M.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

Entrenamiento: definimos talla del batch y número de épocas (iteraciones)

In [ ]:
```
H = M.fit(x_train, y_train, batch_size=16, epochs=10, verbose=0)
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5))
xx = np.arange(1, len(H.history['loss'])+1)
ax = axes[0]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('loss')
ax.set_xticks(xx); ax.plot(xx, H.history['loss'], color='b', marker='s')
ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy')
ax.set_xticks(xx); ax.plot(xx, H.history['accuracy'], color='b', marker='s');
```



Evaluación:

In [ ]:
```
score = M.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.1%}')
```

```
Loss: 0.1051
Precisión: 98.1%
```

6 Ejercicio: Fashion-MNIST

Ejercicio: realiza un experimento similar al de MNIST con Fashion-MNIST

Inicialización: librerías y semilla para la generación de números aleatorios

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras
keras.utils.set_random_seed(23)
```

Lectura de Fashion-MNIST:

In [ ]:
```
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()
print(f'Lectura: train {x_train.shape} {y_train.shape} test {x_test.shape} {y_test.shape}')

input_dim = x_train.shape[1] * x_train.shape[2]
x_train = x_train.reshape(-1, input_dim).astype("float32")
x_test = x_test.reshape(-1, input_dim).astype("float32")
print(f'Reformato 1: train {x_train.shape} {y_train.shape} test {x_test.shape} {y_test.shape}')

x_train_max = np.max(x_train)
x_train /= x_train_max
x_test /= x_train_max
print(f'Normalización [0,1]: max = {x_train_max}')

num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
print(f'Reformato 2: train {x_train.shape} {y_train.shape} test {x_test.shape} {y_test.shape}')
```

```
Lectura: train (60000, 28, 28) (60000,) test (10000, 28, 28) (10000,)
Reformato 1: train (60000, 784) (60000,) test (10000, 784) (10000,)
Normalización [0,1]: max = 255.0
Reformato 2: train (60000, 784) (60000, 10) test (10000, 784) (10000, 10)
```

Definición del modelo: MLP con una capa oculta de ReLUs y capa de salida softmax

In [ ]:
```
M = keras.Sequential([keras.Input(shape=(input_dim,)),
    keras.layers.Dense(800, activation='relu'),
    keras.layers.Dense(num_classes, activation='softmax')])
M.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 800)	628,000
dense_1 (Dense)	(None, 10)	8,010

Total params: 636,010 (2.43 MB)

Trainable params: 636,010 (2.43 MB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
print(f'Número de parámetros de la capa oculta: {784*800 + 800}')
print(f'Número de parámetros de la capa de salida: {800*10 + 10}')
```

```
Número de parámetros de la capa oculta: 628000
Número de parámetros de la capa de salida: 8010
```

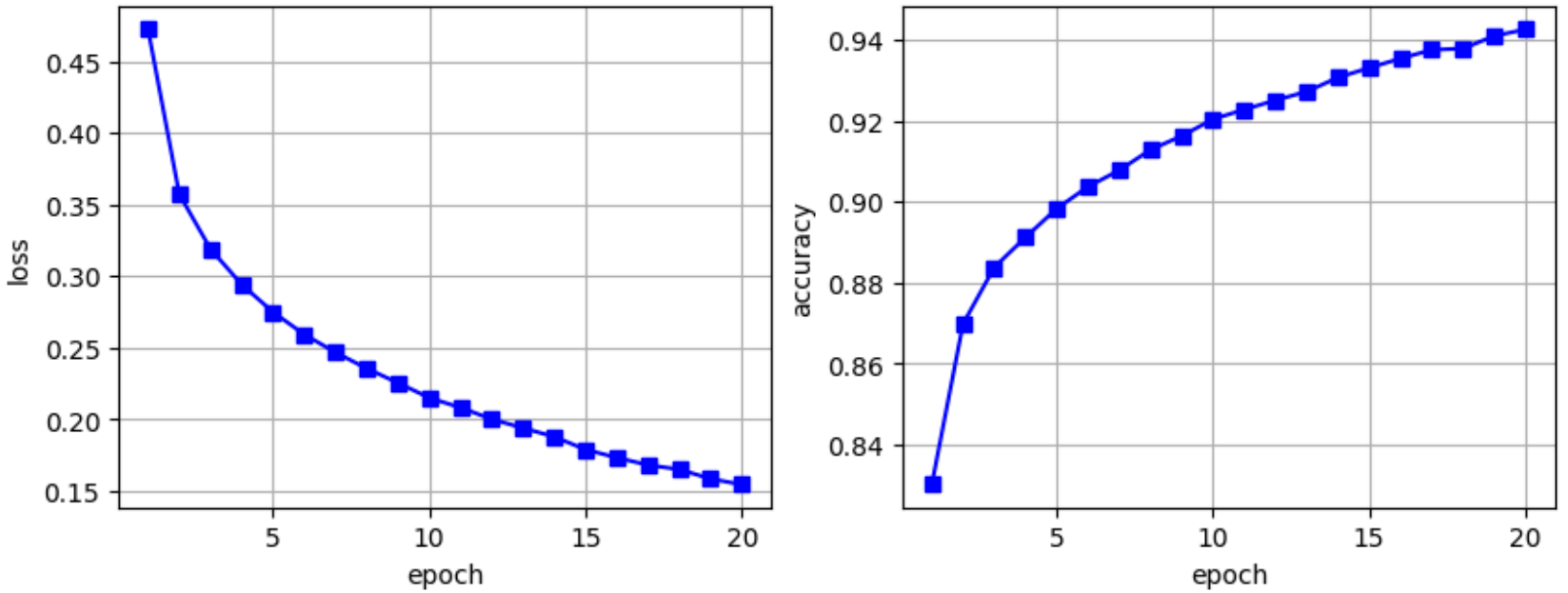
Compilación del modelo: definimos pérdida, optimizador y métrica

In [ ]:
```
M.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

Entrenamiento: definimos talla del batch y número de épocas (iteraciones)

In [ ]:
```
H = M.fit(x_train, y_train, batch_size=16, epochs=20, verbose=0)
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(10, 3.5))
xx = np.arange(1, len(H.history['loss'])+1)
ax = axes[0]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('loss')
ax.plot(xx, H.history['loss'], color='b', marker='s')
ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy')
ax.plot(xx, H.history['accuracy'], color='b', marker='s');
```



Evaluación:

In [ ]:
```
score = M.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.1%}')
```

```
Loss: 0.4734
Precisión: 88.0%
```