TL04 Ajuste del entrenamiento

Índice

1 Ajuste del learning rate y batch size

Learning rate: principal hiperparámetro a ajustar en descenso por gradiente estocástico (SGD)

- Si es demasiado pequeño, SGD converge muy lentamente
- Si es demasiado grande, SGD puede no converger
- Para empezar, es buena idea ajustar un valor constante que obtenga resultados comparativamente buenos
- También conviene añadir terminación temprana para evitar que los experimentos se alarguen demasiado
- Terminación temprana requerirá que monitoricemos una métrica en validación
- La "paciencia" de terminación temprana puede verse como el número mínimo de épocas que queremos ejecutar
- Un learning rate algo grande puede verse como una forma de regularización que evita caer en mínimos estrechos

Batch size: otro hiperparámetro a ajustar muy importante

- Si es demasiado pequeño, el objetivo es muy ruidoso y SGD puede mostrar un comportamiento errático
- Si es demasiado grande, el objetivo es poco ruidoso y SGD tiende a caer en un mínimo estrecho
- Un batch size algo pequeño puede verse como una forma de regularización que evita caer en mínimos estrechos

MNIST: resultados previos

- MLP inicial: MLP con una capa oculta de 800 RELUs, batch size 16, 10 épocas; 98.1% en test
- Mejor arquitectura: una capa oculta de 800 RELUs, 98.2% en val, 98.2% en test (98.2% modelo val)

Inicialización: librerías, semilla, lectura de MNIST y partición train-val-test

In [ ]:
```
import numpy as np; import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras; import keras_tuner
keras.utils.set_random_seed(23); input_dim = 784; num_classes = 10
(x_train_val, y_train_val), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train_val = x_train_val.reshape(-1, input_dim).astype("float32") / 255.0
x_test = x_test.reshape(-1, input_dim).astype("float32") / 255.0
y_train_val = keras.utils.to_categorical(y_train_val, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
x_train = x_train_val[:-10000]; x_val = x_train_val[-10000:]
y_train = y_train_val[:-10000]; y_val = y_train_val[-10000:]
```

MyHyperModel: tras pruebas informales, exploramos learning rate próximo a 0.0017 y batch size de 64, 128 o 256

In [ ]:
```python
class MyHyperModel(keras_tuner.HyperModel):
    def build(self, hp):
        M = keras.Sequential()
        M.add(keras.Input(shape=(784,)))
        M.add(keras.layers.Dense(units=800, activation='relu'))
        M.add(keras.layers.Dense(10, activation='softmax'))
        learning_rate = hp.Float("lr", min_value=0.0015, max_value=0.0019)
        opt = keras.optimizers.Adam(learning_rate=learning_rate)
        M.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
        return M
    def fit(self, hp, M, x, y, xy_val, **kwargs):
        bs = hp.Int("batch_size", 64, 256, step=2, sampling="log")
        early_cb = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=10, min_delta=1e-5)
        kwargs['callbacks'].append(early_cb)
        return M.fit(x, y, batch_size=bs, epochs=100, validation_data=xy_val, **kwargs)
```

Experimento: solo hacemos 3 experimentos para facilitar computacionalmente su reproducción

In [ ]:
```
tuner = keras_tuner.BayesianOptimization(
    MyHyperModel(), objective="val_accuracy", max_trials=3,
    overwrite=True, directory="/tmp", project_name="MNIST")
```

In [ ]:
```
tuner.search(x_train, y_train, (x_val, y_val))
```

```
Trial 3 Complete [00h 00m 54s]
val_accuracy: 0.9829000234603882

Best val_accuracy So Far: 0.9847000241279602
Total elapsed time: 00h 03m 41s
```

In [ ]:
```
tuner.results_summary(num_trials=1)
```

```
Results summary
Results in /tmp/MNIST
Showing 1 best trials
Objective(name="val_accuracy", direction="max")

Trial 1 summary
Hyperparameters:
lr: 0.0016787335100293622
batch_size: 256
Score: 0.9847000241279602
```

In [ ]:
```
best = tuner.get_best_models(num_models=1)[0]
score = best.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.2%}')
```

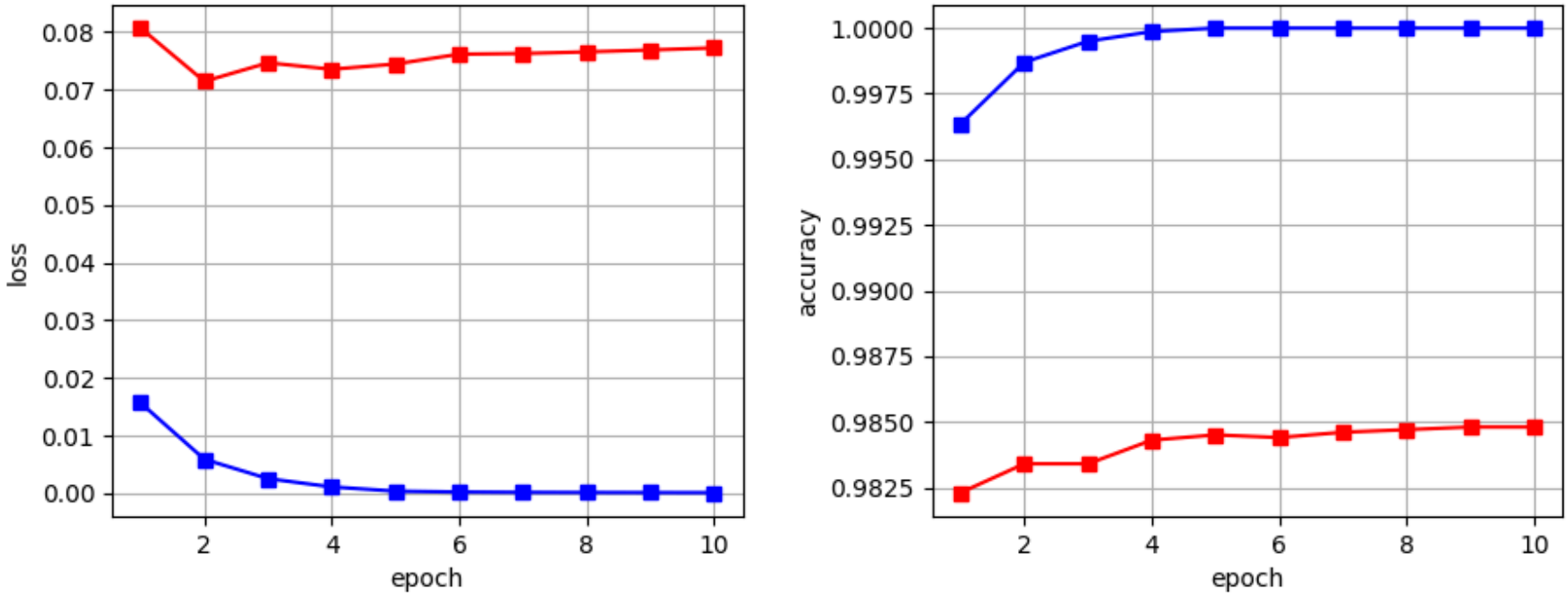
```
Loss: 0.08457
Precisión: 98.45%
```

Experimento (cont.): iteraciones adicionales con train_val y evaluación en test

In [ ]:
```
opt = keras.optimizers.Adam(learning_rate=0.001)
best.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
H = best.fit(x_train_val, y_train_val, batch_size=256, epochs=10, validation_data=(x_test, y_test), verbose=0)
score = best.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.2%}')
```

```
Loss: 0.07724
Precisión: 98.48%
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5))
fig.tight_layout(); plt.subplots_adjust(wspace=0.3)
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.plot(xx, H.history['val_loss'], color='r', 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')
ax.plot(xx, H.history['val_accuracy'], color='r', marker='s');
```



2 Planificadores

Planificador: función para modificar el learning rate a medida que SGD avanza

- Se han propuesto numerosos heurísticos como alternativa al learning rate constante
- El learning rate se suele decrementar paulatinamente para aproximar bien un mínimo del objetivo
- Algunos heurísticos establecen uno o más ciclos de aumento-decremento como forma de regularización
- Conviene añadir terminación temprana para evitar que los experimentos se alarguen demasiado

ReduceLROnPlateau: https://keras.io/api/callbacks/reduce_lr_on_plateau

- Planificador estándar que combina caída escalonada con monitorización de una métrica en validación
- El learning se suele reducir por un factor de 2 (0.5) a 10 (0.1) cuando se "agota la paciencia"
- Como en terminación temprana, la "paciencia" puede verse como el número mínimo de épocas a ejecutar
- Algunos parámetros relevantes con sus valores por omisión:
  - `monitor="val_loss"`: métrica a monitorizar
  - `factor=0.1`: factor de reducción cuando se agota la paciencia
  - `patience=10`: paciencia
  - `min_delta=0.0001`: variación mínima de la métrica para considerarla significativa
  - `min_lr=0.0`: cota inferior del learning rate

MNIST: resultados previos

- MLP inicial: MLP con una capa oculta de 800 RELUs, batch size 16, 10 épocas; 98.1% en test
- Mejor arquitectura: una capa oculta de 800 RELUs, 98.2% en val, 98.2% en test (98.2% modelo val)
- Learning rate y batch size: ajustados a 0.00168 y 256; 98.5% en val, 98.5% en test (98.5% modelo val)

Inicialización: librerías, semilla, lectura de MNIST y partición train-val-test

In [ ]:
```
import numpy as np; import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras; import keras_tuner
keras.utils.set_random_seed(23); input_dim = 784; num_classes = 10
(x_train_val, y_train_val), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train_val = x_train_val.reshape(-1, input_dim).astype("float32") / 255.0
x_test = x_test.reshape(-1, input_dim).astype("float32") / 255.0
y_train_val = keras.utils.to_categorical(y_train_val, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
x_train = x_train_val[:-10000]; x_val = x_train_val[-10000:]
y_train = y_train_val[:-10000]; y_val = y_train_val[-10000:]
```

MyHyperModel: exploramos factor de reducción y paciencia (doble para terminación temprana)

In [ ]:
```python
class MyHyperModel(keras_tuner.HyperModel):
    def build(self, hp):
        M = keras.Sequential()
        M.add(keras.Input(shape=(784,)))
        M.add(keras.layers.Dense(units=800, activation='relu'))
        M.add(keras.layers.Dense(10, activation='softmax'))
        opt = keras.optimizers.Adam(learning_rate=0.00168)
        M.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
        return M
    def fit(self, hp, M, x, y, xy_val, **kwargs):
        factor = hp.Float("factor", min_value=0.1, max_value=0.5)
        patience = hp.Choice("patience", [2, 5, 10])
        reduce_cb = keras.callbacks.ReduceLROnPlateau(
            monitor='val_accuracy', factor=factor, patience=patience, min_delta=1e-4, min_lr=1e-5)
        early_cb = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=2*patience, min_delta=1e-5)
        kwargs['callbacks'].extend([reduce_cb, early_cb])
        return M.fit(x, y, batch_size=256, epochs=100, validation_data=xy_val, **kwargs)
```

Experimento: exploración y evaluación en test del mejor modelo en validación

In [ ]:
```
tuner = keras_tuner.BayesianOptimization(
    MyHyperModel(), objective="val_accuracy", max_trials=10,
    overwrite=True, directory="/tmp", project_name="MNIST")
```

In [ ]:
```
tuner.search(x_train, y_train, (x_val, y_val))
```

```
Trial 10 Complete [00h 01m 24s]
val_accuracy: 0.9846000075340271

Best val_accuracy So Far: 0.9850000143051147
Total elapsed time: 00h 08m 52s
```

In [ ]:
```
tuner.results_summary(num_trials=1)
```

```
Results summary
Results in /tmp/MNIST
Showing 1 best trials
Objective(name="val_accuracy", direction="max")

Trial 02 summary
Hyperparameters:
factor: 0.37871750399265836
patience: 10
Score: 0.9850000143051147
```

In [ ]:
```
best = tuner.get_best_models(num_models=1)[0]
score = best.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.2%}')
```

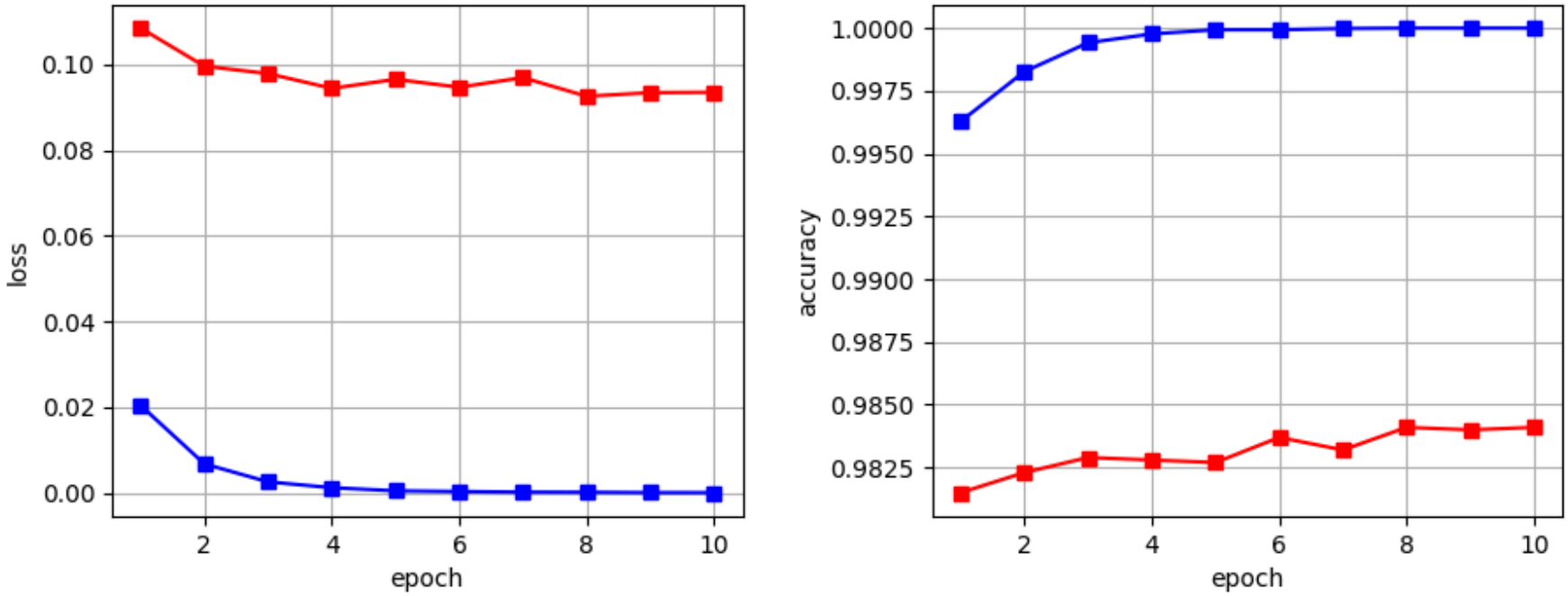
```
Loss: 0.1054
Precisión: 98.35%
```

Experimento (cont.): iteraciones adicionales con train_val y evaluación en test

In [ ]:
```
opt = keras.optimizers.Adam(learning_rate=0.001)
best.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
H = best.fit(x_train_val, y_train_val, batch_size=256, epochs=10, validation_data=(x_test, y_test), verbose=0)
score = best.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.2%}')
```

```
Loss: 0.09352
Precisión: 98.41%
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(9, 3.5))
fig.tight_layout(); plt.subplots_adjust(wspace=0.3)
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.plot(xx, H.history['val_loss'], color='r', 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')
ax.plot(xx, H.history['val_accuracy'], color='r', marker='s');
```



MNIST: resumen de resultados

- MLP inicial: MLP con una capa oculta de 800 RELUs, batch size 16, 10 épocas; 98.1% en test
- Mejor arquitectura: una capa oculta de 800 RELUs, 98.2% en val, 98.2% en test (98.2% modelo val)
- Learning rate y batch size: ajustados a 0.00168 y 256; 98.5% en val, 98.5% en test (98.5% modelo val)
- ReduceLROnPlateau: factor 0.3787 y paciencia 10; 98.5% en val, 98.4% en test (98.4% modelo val)

3 Ejercicio: Fashion-MNIST

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

3.1 Inicialización

Inicialización: librerías, semilla, lectura de MNIST y partición train-val-test

In [ ]:
```
import numpy as np; import matplotlib.pyplot as plt
import os; os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import keras; import keras_tuner
keras.utils.set_random_seed(23); input_dim = 784; num_classes = 10
(x_train_val, y_train_val), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()
x_train_val = x_train_val.reshape(-1, input_dim).astype("float32") / 255.0
x_test = x_test.reshape(-1, input_dim).astype("float32") / 255.0
y_train_val = keras.utils.to_categorical(y_train_val, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
x_train = x_train_val[:-10000]; x_val = x_train_val[-10000:]
y_train = y_train_val[:-10000]; y_val = y_train_val[-10000:]
```

3.2 Ajuste del learning rate y batch size

Resultados previos:

- MLP inicial: MLP con una capa oculta de 800 RELUs, batch size 16, 20 épocas; 88.0% en test
- Mejor arquitectura: una capa oculta de 800 RELUs, 89.0% en val, 88.3% en test (88.0% modelo val)

MyHyperModel: tras pruebas informales, exploramos learning rate próximo a 0.00016 y batch size de 128 o 256

In [ ]:
```python
class MyHyperModel(keras_tuner.HyperModel):
    def build(self, hp):
        M = keras.Sequential()
        M.add(keras.Input(shape=(784,)))
        M.add(keras.layers.Dense(units=800, activation='relu'))
        M.add(keras.layers.Dense(10, activation='softmax'))
        # learning_rate = hp.Float("lr", min_value=1e-5, max_value=0.01, step=2, sampling="log")
        # lr: 0.00016 batch_size: 256 Score: 0.8977
        learning_rate = hp.Float("lr", min_value=0.00006, max_value=0.00026)
        opt = keras.optimizers.Adam(learning_rate=learning_rate)
        M.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
        return M
    def fit(self, hp, M, x, y, xy_val, **kwargs):
        bs = hp.Int("batch_size", 128, 256, step=2, sampling="log")
        early_cb = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=10, min_delta=1e-5)
        kwargs['callbacks'].append(early_cb)
        return M.fit(x, y, batch_size=bs, epochs=100, validation_data=xy_val, **kwargs)
```

Experimento: solo hacemos 3 experimentos para facilitar computacionalmente su reproducción

In [ ]:
```
tuner = keras_tuner.BayesianOptimization(
    MyHyperModel(), objective="val_accuracy", max_trials=3,
    overwrite=True, directory="/tmp", project_name="Fashion-MNIST")
```

In [ ]:
```
tuner.search(x_train, y_train, (x_val, y_val))
```

```
Trial 3 Complete [00h 01m 20s]
val_accuracy: 0.8920999765396118

Best val_accuracy So Far: 0.8960000276565552
Total elapsed time: 00h 04m 08s
```

In [ ]:
```
tuner.results_summary(num_trials=1)
```

```
Results summary
Results in /tmp/Fashion-MNIST
Showing 1 best trials
Objective(name="val_accuracy", direction="max")

Trial 1 summary
Hyperparameters:
lr: 0.00014936675501468113
batch_size: 256
Score: 0.8960000276565552
```

In [ ]:
```
best = tuner.get_best_models(num_models=1)[0]
score = best.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.2%}')
```

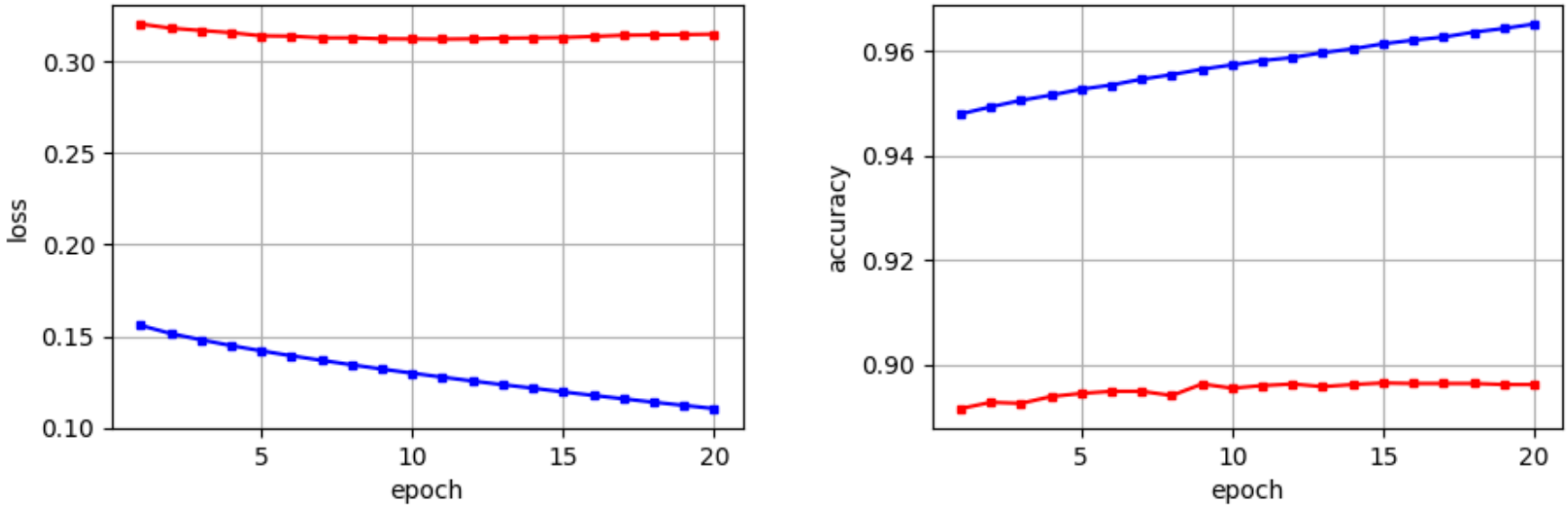
```
Loss: 0.327
Precisión: 89.08%
```

Experimento (cont.): iteraciones adicionales con train_val y evaluación en test

In [ ]:
```
opt = keras.optimizers.Adam(learning_rate=0.00015)
best.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
H = best.fit(x_train_val, y_train_val, batch_size=256, epochs=20, validation_data=(x_test, y_test), verbose=0)
score = best.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.2%}')
```

```
Loss: 0.3103
Precisión: 89.77%
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(9, 3))
fig.tight_layout(); plt.subplots_adjust(wspace=0.3)
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', markersize=3)
ax.plot(xx, H.history['val_loss'], color='r', marker='s', markersize=3)
ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy')
ax.plot(xx, H.history['accuracy'], color='b', marker='s', markersize=3)
ax.plot(xx, H.history['val_accuracy'], color='r', marker='s', markersize=3);
```



3.3 Planificadores

Resultados previos:

- MLP inicial: MLP con una capa oculta de 800 RELUs, batch size 16, 20 épocas; 88.0% en test
- Mejor arquitectura: una capa oculta de 800 RELUs, 89.0% en val, 88.3% en test (88.0% modelo val)
- Learning rate y batch size: ajustados a 0.00015 y 256; 89.6% en val, 89.8% en test (89.1% modelo val)

MyHyperModel: exploramos factor de reducción y paciencia (doble para terminación temprana)

In [ ]:
```python
class MyHyperModel(keras_tuner.HyperModel):
    def build(self, hp):
        M = keras.Sequential()
        M.add(keras.Input(shape=(784,)))
        M.add(keras.layers.Dense(units=800, activation='relu'))
        M.add(keras.layers.Dense(10, activation='softmax'))
        opt = keras.optimizers.Adam(learning_rate=0.00015)
        M.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
        return M
    def fit(self, hp, M, x, y, xy_val, **kwargs):
        factor = hp.Float("factor", min_value=0.1, max_value=0.5)
        patience = hp.Choice("patience", [2, 5, 10])
        reduce_cb = keras.callbacks.ReduceLROnPlateau(
            monitor='val_accuracy', factor=factor, patience=patience, min_delta=1e-4, min_lr=1e-5)
        early_cb = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=2*patience, min_delta=1e-5)
        kwargs['callbacks'].extend([reduce_cb, early_cb])
        return M.fit(x, y, batch_size=256, epochs=100, validation_data=xy_val, **kwargs)
```

Experimento: exploración y evaluación en test del mejor modelo en validación

In [ ]:
```
tuner = keras_tuner.BayesianOptimization(
    MyHyperModel(), objective="val_accuracy", max_trials=10,
    overwrite=True, directory="/tmp", project_name="Fashion-MNIST")
```

In [ ]:
```
tuner.search(x_train, y_train, (x_val, y_val))
```

```
Trial 10 Complete [00h 01m 55s]
val_accuracy: 0.9003999829292297

Best val_accuracy So Far: 0.9003999829292297
Total elapsed time: 00h 15m 05s
```

In [ ]:
```
tuner.results_summary(num_trials=1)
```

```
Results summary
Results in /tmp/Fashion-MNIST
Showing 1 best trials
Objective(name="val_accuracy", direction="max")

Trial 03 summary
Hyperparameters:
factor: 0.317454502932933
patience: 5
Score: 0.9003999829292297
```

In [ ]:
```
best = tuner.get_best_models(num_models=1)[0]
score = best.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.2%}')
```

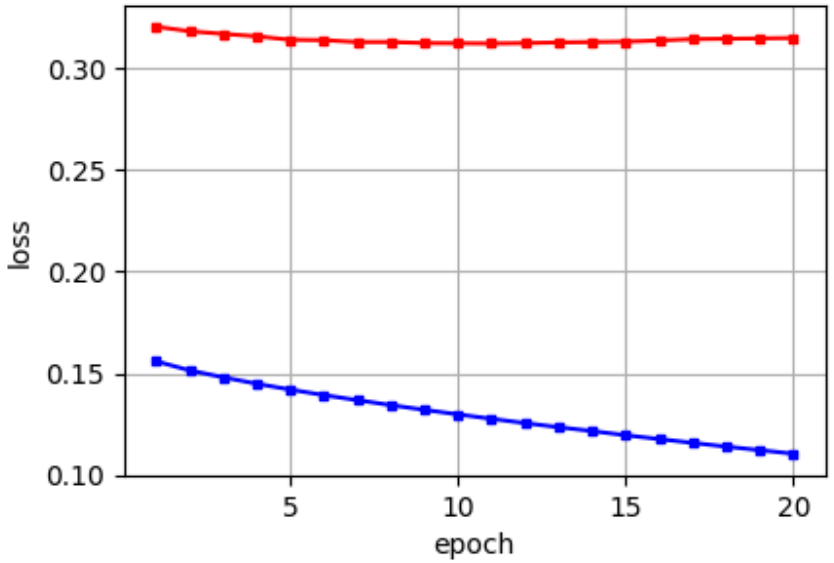
```
Loss: 0.3139
Precisión: 89.45%
```

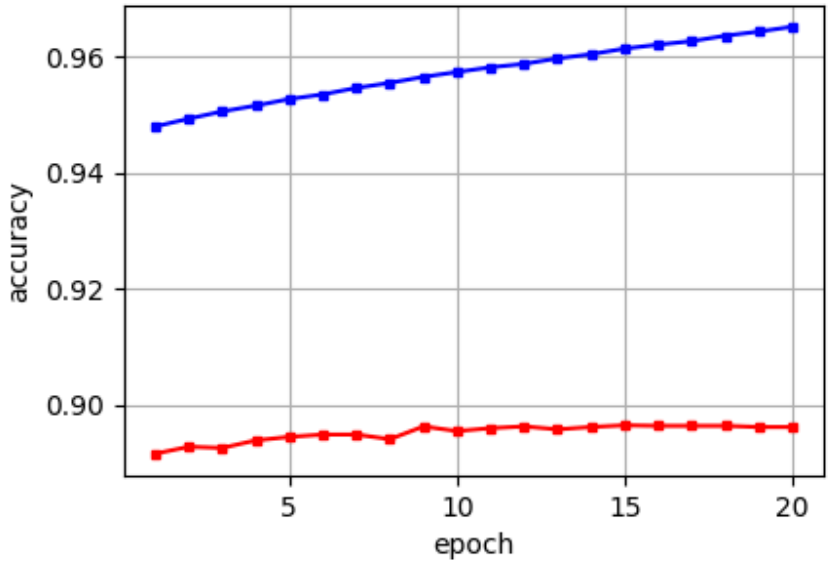
Experimento (cont.): iteraciones adicionales con train_val y evaluación en test

In [ ]:
```
opt = keras.optimizers.Adam(learning_rate=0.00015)
best.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
H = best.fit(x_train_val, y_train_val, batch_size=256, epochs=20, validation_data=(x_test, y_test), verbose=0)
score = best.evaluate(x_test, y_test, verbose=0)
print(f'Loss: {score[0]:.4}\nPrecisión: {score[1]:.2%}')
```

```
Loss: 0.3151
Precisión: 89.62%
```

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(9, 3))
fig.tight_layout(); plt.subplots_adjust(wspace=0.3)
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', markersize=3)
ax.plot(xx, H.history['val_loss'], color='r', marker='s', markersize=3)
ax = axes[1]; ax.grid(); ax.set_xlabel('epoch'); ax.set_ylabel('accuracy')
ax.plot(xx, H.history['accuracy'], color='b', marker='s', markersize=3)
ax.plot(xx, H.history['val_accuracy'], color='r', marker='s', markersize=3);
```





Resumen de resultados:

- MLP inicial: MLP con una capa oculta de 800 RELUs, batch size 16, 20 épocas; 88.0% en test
- Mejor arquitectura: una capa oculta de 800 RELUs, 89.0% en val, 88.3% en test (88.0% modelo val)
- Learning rate y batch size: ajustados a 0.00015 y 256; 89.6% en val, 89.8% en test (89.1% modelo val)
- ReduceLROnPlateau: factor 0.32 y paciencia 5; 90.0% en val, 89.6% en test (89.5% modelo val)