TL05 Inicialización de los pesos de una red

Índice

1 Inicialización de los pesos de una red

Inicializadores de los pesos de una red:

- **API:** https://keras.io/api/layers/initializers
- **Uso:** heurísticos para inicializar (aleatoriamente) los pesos de las capas
- **Opciones:** RandomNormal, RandomUniform, TruncatedNormal, Zeros, Ones, GlorotNormal, GlorotUniform, etc.
- **Inicializador por omisión de una capa lineal (Dense):** GlorotUniform

Ejemplo: la sección 2 incluye un ejemplo de inicialización para MNIST

Examen: la sección 3 describe el ejercicio a realizar, similar al ejemplo, pero con Fashion-MNIST

2 MNIST con inicialización RandomNormal

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)

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 inicialización RandomNormal con desviación estándar entre 0.01 y 0.10

In [ ]:
```
class MyHyperModel(keras_tuner.HyperModel):
    def build(self, hp):
        M = keras.Sequential()
        M.add(keras.Input(shape=(784,)))
        stddev = hp.Float("stddev", min_value=0.01, max_value=0.10)
        kernel_initializer = keras.initializers.RandomNormal(stddev=stddev)
        M.add(keras.layers.Dense(units=800, activation='relu', kernel_initializer=kernel_initializer))
        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 = 0.3787; patience = 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, executions_per_trial=1,
    overwrite=True, directory="/tmp", project_name="MNIST")
```

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

```
Trial 10 Complete [00h 00m 18s]
val_accuracy: 0.9842000007629395

Best val_accuracy So Far: 0.9850999712944031
Total elapsed time: 00h 02m 37s
```

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

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

Trial 03 summary
Hyperparameters:
stddev: 0.036970858597258155
Score: 0.9850999712944031
```

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: 0.07737
Precisión: 98.45%
```

Conclusión: precisión en test similar a la que teníamos

3 Fashion-MNIST con inicialización RandomNormal

Fashion-MNIST:

- 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)

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

- Emplea la arquitectura e hiperparámetros ajustados previamente (indicados arriba)
- Añade una breve conclusión al final sobre la bondad relativa de los resultados

Inicialización: librerías, semilla, lectura de Fashion-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:]
```

MyHyperModel: exploramos inicialización RandomNormal con desviación estándar entre 0.01 y 0.10

In [ ]:
```python
class MyHyperModel(keras_tuner.HyperModel):
    def build(self, hp):
        M = keras.Sequential()
        M.add(keras.Input(shape=(784,)))
        stddev = hp.Float("stddev", min_value=0.01, max_value=0.10)
        kernel_initializer = keras.initializers.RandomNormal(stddev=stddev)
        M.add(keras.layers.Dense(units=800, activation='relu', kernel_initializer=kernel_initializer))
        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 = 0.32; patience = 5
        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, executions_per_trial=1,
    overwrite=True, directory="/tmp", project_name="Fashion-MNIST")
```

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

```
Trial 10 Complete [00h 00m 30s]
val_accuracy: 0.9000999927520752

Best val_accuracy So Far: 0.9010000228881836
Total elapsed time: 00h 04m 22s
```

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 02 summary
Hyperparameters:
stddev: 0.0536326844595774
Score: 0.9010000228881836
```

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: 0.3152
Precisión: 89.44%
```

Conclusión: precisión en test similar a la que teníamos