Clase 103 — Keras Functional API y Subclassing

Parte: 2 — Deep Learning · Fuente: Géron, cap. 10 § Building Complex Models Using the Functional API y § Building Dynamic Models Using the Subclassing API. Duración estimada: 70 min.

Clase 103 — Keras Functional API y Subclassing

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 10 § Building Complex Models Using the Functional API y § Building Dynamic Models Using the Subclassing API. Duración estimada: 70 min.*

Objetivo

Construir modelos con topologías no lineales —skip connections, multi-input, multi-output, capas compartidas— usando la Functional API (estilo "grafo de capas"), y modelos con flujo de control dinámico (loops, ifs) usando Subclassing (estilo class MyModel(Model) con call()). Saber elegir entre las tres APIs según el caso.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Construir un modelo Wide & Deep (clásico de Cheng et al. 2016) con Functional API.
- Construir un modelo multi-output con dos Dense finales y dos losses.
- Implementar un MyResBlock con Subclassing que tiene una skip connection.
- Reconocer el trade-off: Sequential (simple) → Functional (la mayoría de los casos) → Subclassing (cuando hace falta control dinámico).
- Convertir un modelo Functional en JSON / cargar con model_from_json (útil para serialización separada de pesos).

Temas

- Functional API: x = layer(prev_x); al final Model(inputs=[...], outputs=[...]).
- Multi-input / multi-output: pasar listas o dicts a inputs= / outputs=.
- Capas compartidas (siamese networks): aplicar la misma instancia de capa a dos entradas distintas.
- Subclassing: heredar de keras.Model, definir __init__ (capas) y call(self, inputs, training=False).
- Cuándo subclassing: control de flujo dinámico, modelos imperativos estilo PyTorch.

Definiciones y características

- Functional API: cada capa se llama como función output = LayerInstance()(input); permite cualquier DAG. Visualizable con keras.utils.plot_model.
- Subclassing: Model custom donde call() define el forward pass arbitrariamente. Más flexible pero más difícil de serializar y debuggear.
- Skip connection: y = Add()([x, F(x)]) — clave en ResNet (clase 115).
- Capa compartida: shared = Dense(32); a = shared(x1); b = shared(x2). Misma matriz de pesos en ambos.
- call(self, inputs, training=False): el flag training distingue forward de entrenamiento (con dropout activo) vs inferencia.

Dataset / recursos

- California Housing para Wide & Deep.
- Librerías: tensorflow, keras.

Ejercicios

1. Wide & Deep: implementá el modelo de Cheng et al. con Functional API — input → wide path (directo a salida) + deep path (Dense × 2 → salida) → Add() → output.
2. Multi-output: predecir simultáneamente precio (regresión) Y rango de precio (clasificación 3 clases) sobre California Housing. Compilar con dict de losses.
3. Capa compartida (siamese): dos imágenes de entrada → mismo encoder Dense(64) aplicado a ambas → concatenar embeddings → clasificación "same/different".
4. ResBlock con Subclassing: implementá una clase ResBlock(Layer) con dos Dense y skip connection. Usala en un modelo Sequential.
5. plot_model: graficá el modelo Wide & Deep con keras.utils.plot_model(model, show_shapes=True). Identificá visualmente las dos rutas.

Homework verificable

Implementar un modelo multi-output en California Housing:

1. Predecir median_house_value (regresión, loss MSE).
2. Predecir expensive (binario: above/below mediana, loss BCE).
3. Compartir las 2 primeras capas Dense (representación común) y luego ramificar.
4. Compilar con loss=[mse, bce], loss_weights=[0.7, 0.3].
5. Entrenar y reportar MAE + accuracy.

Criterio de aceptación: el modelo entrena sin errores, MAE razonable (< $50k), accuracy ≥ 0.85; plot_model muestra la rama compartida + dos heads.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ValueError: A merge layer should be called	Add()(x, y) en vez de Add()([x, y]). Fix:
Subclass call() no respeta training y drop	Fix: aceptar training=False como argumento
model.save() falla en subclass	Subclassing requiere implementar get_confi
Multi-output con loss='mse' (un solo strin	Aplica MSE a las dos salidas. Fix: pasar l
Capa creada dentro de call() en cada forwa	Crea pesos nuevos cada vez → no aprende. F

Preguntas frecuentes

¿Functional o Subclassing?

Functional por default. Es más fácil de serializar, visualizar y debuggear. Subclassing solo si necesitás control de flujo dinámico (loops, ifs sobre el input — raro en práctica salvo RNN custom).

¿Puedo mezclar Sequential dentro de Functional?

Sí. inner = Sequential([Dense(64), Dense(32)]); out = inner(x) — Sequential es un Layer válido.

¿Functional es más lento que Sequential?

No, mismo grafo por debajo.

¿Cuándo necesito multi-input?

Cuando tenés modalidades distintas (imagen + metadatos del usuario, texto + features numéricas). Cada modalidad tiene su input y su encoder; al final se concatenan.

¿Add() o Concatenate()?

Add() para skip connections (preserva dimensión). Concatenate() para fusión de modalidades (suma dimensiones).

Referencias

- Géron, cap. 10 — The Functional API y The Subclassing API.
- Cheng et al. (2016), Wide & Deep Learning for Recommender Systems, DLRS.
- Keras — The Functional API.
- Keras — Making new layers and models via subclassing.

Siguiente clase

Clase 104 — Callbacks, TensorBoard, guardar/restaurar modelos

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb