Clase 105 — Keras Tuner (+ Optuna, Ray Tune)

Parte: 2 — Deep Learning · Fuente: Géron, cap. 10 § Fine-Tuning Neural Network Hyperparameters + docs Keras Tuner / Optuna / Ray Tune. Duración estimada: 85 min.

Clase 105 — Keras Tuner (+ Optuna, Ray Tune)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 10 § Fine-Tuning Neural Network Hyperparameters + docs Keras Tuner / Optuna / Ray Tune. Duración estimada: 85 min.*

Objetivo

Hacer hyperparameter tuning sistemático en redes neuronales — buscar n_layers, units, lr, dropout, etc. con estrategias modernas: Random Search, Hyperband y Bayesian Optimization. Conocer las tres herramientas estándar de Python — Keras Tuner (TF-native, simple), Optuna (multi-framework, default industrial) y Ray Tune (distribuido, escalable a clusters).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Definir un model-building function con hiperparámetros declarados vía hp.Int, hp.Float, hp.Choice.
- Lanzar tuners de Keras Tuner: RandomSearch, Hyperband, BayesianOptimization.
- Migrar el mismo problema a Optuna con optuna.create_study(direction='minimize') y trial.suggest_float.
- Lanzar Ray Tune con tune.run para distribuir trials en múltiples GPUs/nodos.
- Comparar las 3 estrategias (Random vs Hyperband vs Bayesian) en términos de eficiencia.

Temas

- ¿Por qué tuning? Los defaults (Adam lr=1e-3, dropout 0.5) rara vez son óptimos.
- Random Search vs Grid Search: Bergstra & Bengio (2012) — random gana casi siempre por el "curse of low effective dimensionality".
- Hyperband (Li et al. 2017): asignar más cómputo a configs prometedoras (successive halving).
- Bayesian Optimization (TPE, GP): construye un modelo del paisaje y elige el siguiente trial inteligentemente.
- Trade-off cómputo vs ganancia: típicamente 50-200 trials para una primera optimización.
- Complemento moderno: Optuna y Ray Tune como alternativas multi-framework.

Versión profundizada — 2026

El tema moderno que vivía como complemento dentro de esta clase ahora tiene clase propia dedicada con patrón completo, ejercicios y homework:

- Clase 052a — Optuna y HPO bayesiano dedicado (ML clásico)
- Clase 095b — Ray Tune: HPO distribuido y a escala

Definiciones y características

- Search space: el rango/conjunto donde busca cada hiperparámetro.
- Trial: una corrida de entrenamiento con una config específica.
- Random Search: muestrea trials al azar del espacio.

- Hyperband: corre muchos trials cortos, mata a los peores, sigue con los buenos por más épocas (successive halving en múltiples brackets).
- Bayesian Optimization / TPE (Tree-structured Parzen Estimator): modela P(config | resultado bueno) y muestrea de ahí.
- Pruner: lógica para matar trials sin terminar (callback en cada época).
- log=True en suggest_float: muestreo log-uniform — esencial para LR.

Dataset / recursos

- Fashion-MNIST como dataset chico para iterar rápido.
- Librerías: keras-tuner (pip install keras-tuner), optuna, ray[tune].

Ejercicios

1. Keras Tuner — RandomSearch: tunear units {32, 64, 128} y lr [1e-4, 1e-2] log con 20 trials. Reportar mejores hiperparámetros y val_accuracy.
2. Keras Tuner — Hyperband: el mismo espacio, 50 trials con Hyperband. Comparar tiempo total vs RandomSearch.
3. Optuna: traducir el espacio a Optuna; correr 50 trials con HyperbandPruner; graficar plot_optimization_history y plot_param_importances.
4. Multi-objective: optimizar simultáneamente val_accuracy (max) y n_params (min) con optuna.create_study(directions=['maximize', 'minimize']). Inspeccionar la Pareto front.
5. Visualización: con Optuna, generar plot_parallel_coordinate(study) y entender qué dimensiones son las más sensibles.

Homework verificable

Tunear un MLP sobre Fashion-MNIST con Optuna:

1. Espacio: n_layers [1, 4], units {32, 64, 128, 256}, dropout [0, 0.5], lr [1e-5, 1e-2] log, optimizer {Adam, SGD}.
2. 100 trials con HyperbandPruner, n_jobs=2.
3. Reportar best_params, best_value, y guardar el modelo final entrenado con esos params.
4. Generar los 3 plots de visualización.

Criterio de aceptación: val_accuracy del mejor modelo ≥ 0.89; plot_param_importances debe mostrar lr y/o units como las más importantes (típico).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
LR muestreado uniforme entre 1e-5 y 1e-2 y	LR debe ser log-uniform. Fix: trial.sugges
Bayesian optimization parece random	Necesita >10-20 trials para que el modelo
Hyperband poda trials demasiado pronto	factor muy alto o max_epochs muy bajo. Fix
optuna se cuelga al usar n_jobs > 1 con Ke	TF tiene problemas con multi-threading. Fi
Reportar el resultado del mejor trial del	Eso es el resultado de búsqueda, no de eva

Preguntas frecuentes

¿Cuántos trials?

Mínimo 20–50 para Random / Hyperband. Bayesian se beneficia de más (100+). Si cada trial cuesta 1h → 100 trials = ~4 días en 1 GPU; con Ray Tune en 4 GPUs, ~1 día.

¿Tuning del LR primero o del resto?

LR primero, siempre. Es de lejos el hiperparámetro más sensible. Después arquitectura, después regularización.

¿Optuna o Keras Tuner para alguien que recién empieza?

Keras Tuner para entender la idea (más simple). Optuna apenas el problema es serio. La transición es rápida.

¿Cómo guardo el resultado del estudio Optuna?

optuna.create_study(study_name='exp1', storage='sqlite:///optuna.db', load_if_exists=True). Persistencia gratis; podés agregar trials después.

¿Ray Tune en una sola máquina vale la pena?

Sí si tenés ≥ 4 cores o ≥ 2 GPUs. Optuna n_jobs también paraleliza pero Ray maneja mejor recursos heterogéneos y crashes.

Referencias

- Géron, cap. 10 — Fine-Tuning Neural Network Hyperparameters.
- Bergstra & Bengio (2012), Random Search for Hyper-Parameter Optimization, JMLR.
- Li et al. (2017), Hyperband, JMLR.
- Akiba et al. (2019), Optuna: A Next-Generation Hyperparameter Optimization Framework, KDD.
- Liaw et al. (2018), Tune: A Research Platform for Distributed Model Selection and Training.
- Keras Tuner docs, Optuna docs, Ray Tune docs.

Siguiente clase

Clase 106 — Ray Tune: HPO distribuido y a escala

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb