Clase 106 — Ray Tune: HPO distribuido y a escala

Parte: 2 — Deep Learning · Fuente: Liaw et al. (2018) Ray Tune + Ray docs. Duración estimada: 75 min.

Clase 106 — Ray Tune: HPO distribuido y a escala

> *Parte: 2 — Deep Learning · Fuente: Liaw et al. (2018) Ray Tune + Ray docs. Duración estimada: 75 min.*

Objetivo

Escalar hyperparameter tuning de DL a cluster con Ray Tune — el framework distribuido que la industria (Uber, Anyscale, OpenAI) usa cuando los trials toman horas y se necesitan decenas en paralelo. Cubrir orquestación, schedulers modernos (ASHA, PBT Population Based Training), integración con W&B, MLflow, y combinación con Optuna como search algorithm.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Definir trainable(config) y reportar progreso con tune.report(loss=...).
- Configurar ASHA (Async Successive Halving) para podar trials malos.
- Aplicar Population Based Training (PBT) para evolución de hyperparams durante training.
- Asignar recursos: resources_per_trial={'cpu': 2, 'gpu': 1}.
- Usar OptunaSearch como search algorithm + Ray Tune como orchestrator.

Temas

- Ray cluster: local vs multi-node.
- Trainable: function-based vs class-based.
- ASHA scheduler — async successive halving.
- PBT — evoluciona hyperparams + checkpoints.
- BOHB — Bayesian opt + Hyperband.
- Loggers: TensorBoard, W&B, MLflow.

Definiciones y características

- Trial: una corrida del trainable.
- Scheduler: decide qué trial para, cuál sigue.
- ASHA: variante asíncrona de Successive Halving — mata trials malos rápido.
- PBT: evolutionary — copia weights de mejores + perturba hyperparams.
- Search algorithm: cómo se eligen configs (random, TPE via Optuna, etc.).

Dataset / recursos

- Fashion-MNIST o cualquier dataset DL.
- Librerías: ray[tune], opcional optuna, lightning.

Ejercicios

1. Trainable básico: train de CNN con metric report each epoch. tune.run(trainable, num_samples=20).
2. ASHA: ASHAScheduler(metric='val_loss', mode='min', max_t=20, grace_period=3). Verificar pruning.
3. PBT: 8 workers, copy + perturb each 5 epochs. Plot evolution de LR.
4. OptunaSearch + ASHA: combination — Optuna sugiere, ASHA poda.
5. Resources: gpus_per_trial=0.5 (fractional GPU sharing).

Homework verificable

Sobre Fashion-MNIST con CNN simple:

1. Tunear LR, batch_size, dropout con Ray Tune.
2. 50 trials, ASHA, OptunaSearch.
3. W&B logging.
4. Reportar mejor config + tiempo total.

Criterio de aceptación: convergencia visible en W&B; pruning de ASHA mata ≥ 30 % trials malos temprano; mejor config supera baseline.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Ray cluster crash con OOM	Workers piden mucha RAM. Fix: resources_pe
PBT requiere checkpointing	Si no lo implementás, PBT falla. Fix: tune
Trainable con loop infinito sin tune.repor	Tune no sabe progreso. Fix: report cada ep
ASHA + grace_period bajo	Mata trials antes de aprender. Fix: grace_
Multi-node sin configurar Ray cluster	Trials van a 1 node. Fix: ray.init(address

Preguntas frecuentes

Ray Tune vs Optuna directo?

Optuna: single-machine, simple. Ray Tune: cluster, mejor orquestación de recursos.

PBT cuándo?

LLM training, modelos largos donde LR decay schedule matter. Para HPO simple, ASHA basta.

Cuántos workers?

Tantos como GPUs disponibles. Local: cpu_count - 2.

Combinar con Lightning?

Sí — ray_lightning para trainer distribuido + Tune para HPO. Más compleja la setup.

Anyscale (Ray managed)?

Cloud service from Ray creators. Para producción serio sin manejar infra propia.

Referencias

- Liaw et al. (2018), Tune: A Research Platform for Distributed Model Selection and Training.

- Li et al. (2018), ASHA.
- Jaderberg et al. (2017), Population Based Training, DeepMind.
- Ray Tune docs.

Siguiente clase

Clase 107 — Vanishing/exploding gradients

Apéndice: notebook (primer bloque)

Ray puede ser pesado de instalar en Windows. Usamos try/except y caemos a Optuna como fallback (mismo concepto).

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# Función objetivo de juguete: paraboloide 2D con mínimo desplazado en (2, -1)
def objective(config):
    x, y = config['x'], config['y']
    # noise simula stochasticidad de entrenamiento
    noise = np.random.normal(0, 0.05)
    return (x - 2)2 + (y + 1)2 + noise

print('óptimo conocido: (x=2, y=-1) → loss=0')
```

Archivos complementarios

- notebook.ipynb