Parte 2 — Deep Learning — Keras, TensorFlow, Transformers, RL y Despliegue

75 clases · Parte 2 del programa

Parte 2 — Deep Learning — Keras, TensorFlow, Transformers, RL y Despliegue

> *75 clases · bundle consolidado del currículo v3.*

Índice de clases

- Clase 135 — Clase 135 — RNNs: neuronas recurrentes, BPTT
- Clase 136 — Clase 136 — Forecasting de series con RNN
- Clase 137 — Clase 137 — LSTM, GRU
- Clase 138 — Clase 138 — 1D CNNs y WaveNet
- Clase 139 — Clase 139 — Generación de texto char-RNN
- Clase 140 — Clase 140 — Análisis de sentimiento
- Clase 141 — Clase 141 — Encoder-Decoder para traducción
- Clase 142 — Clase 142 — Mecanismos de atención
- Clase 143 — Clase 143 — Transformers: arquitectura, BERT, GPT (+ Flash Attention, RoPE, GQA)
- Clase 144 — Clase 144 — Flash Attention v2/v3, RoPE, GQA: el motor de los LLMs modernos
- Clase 145 — Clase 145 — Hugging Face Transformers (uso práctico)
- Clase 146 — Clase 146 — CLIP, SigLIP: multimodal embeddings (visión + texto)
- Clase 147 — Clase 147 — Whisper: ASR, transcripción, traducción de audio
- Clase 148 — Clase 148 — LLMs aplicados: fine-tuning, prompting (+ LoRA / QLoRA, DPO, vLLM)
- Clase 149 — Clase 149 — LoRA / QLoRA: fine-tuning eficiente de LLMs
- Clase 150 — Clase 150 — DPO y RLHF: alineamiento de LLMs
- Clase 151 — Clase 151 — vLLM y TGI: serving de LLMs en producción
- Clase 152 — Clase 152 — RAG básico y embeddings (+ hybrid search, re-ranking, MCP)
- Clase 153 — Clase 153 — MCP (Model Context Protocol): herramientas y datos para LLMs
- Clase 154 — Clase 154 — Agentes: tool use, ReAct, multi-agent
- Clase 155 — Clase 155 — LLM Evaluation: MMLU, MT-Bench, LLM-as-judge, evals propios
- Clase 156 — Clase 156 — Autoencoders: undercomplete, stacked, denoising, sparse
- Clase 157 — Clase 157 — Variational Autoencoders (VAE)
- Clase 158 — Clase 158 — GANs: DCGAN, Progressive GAN, StyleGAN
- Clase 159 — Clase 159 — Modelos de difusión (+ Stable Diffusion XL, ControlNet, LCM)
- Clase 160 — Clase 160 — Stable Diffusion XL + ControlNet en profundidad
- Clase 161 — Clase 161 — RL: aprendizaje por recompensa, Gymnasium (Farama)
- Clase 162 — Clase 162 — Policy gradients
- Clase 163 — Clase 163 — Markov Decision Processes
- Clase 164 — Clase 164 — TD Learning, Q-Learning, Deep Q-Networks
- Clase 165 — Clase 165 — RL moderno: A3C, PPO, SAC (vista general)
- Clase 166 — Clase 166 — TF Serving + gRPC (+ ONNX, TensorRT, vLLM/TGI)
- Clase 167 — Clase 167 — ONNX y ONNX Runtime: portabilidad e inference optimizada
- Clase 168 — Clase 168 — Despliegue en Vertex AI
- Clase 169 — Clase 169 — TF Lite (mobile/embedded)
- Clase 170 — Clase 170 — TensorFlow.js (navegador)
- Clase 171 — Clase 171 — Aceleración con GPU
- Clase 172 — Clase 172 — Entrenamiento multi-dispositivo, tf.distribute
- Clase 173 — Clase 173 — JAX y Flax: el stack moderno de Google para DL
- Clase 174 — Clase 174 — Entrenamiento a escala con Vertex AI

Clase 100 — Clase 100 — Perceptrón, MLP y backpropagation

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 10 § From Biological to Artificial Neurons. Duración estimada: 80 min.*

Objetivo

Que el alumno entienda la unidad fundamental del Deep Learning: la neurona artificial (perceptrón de Rosenblatt 1957), por qué un solo perceptrón no puede aprender XOR, cómo el MLP (Multi-Layer Perceptron) lo resuelve apilando capas con activaciones no lineales, y cómo backpropagation (regla de la cadena) permite calcular gradientes en cualquier grafo computacional — el algoritmo que destrabó el Deep Learning moderno.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Implementar a mano un perceptrón (y = step(w·x + b)) y mostrar por qué no separa XOR.
- Construir un MLP con keras.Sequential([Dense(...), Dense(...)]) y entrenarlo sobre un dataset simple.
- Explicar forward pass (compute layer by layer) y backward pass (propagar gradientes con la regla de la cadena).
- Calcular a mano los gradientes para un MLP de 2 capas con una sola muestra.
- Reconocer que la diferenciación automática (autograd) hace innecesario derivar a mano para modelos arbitrarios.

Temas

#	Tema	Por qué importa
1	Perceptrón clásico (Rosenblatt 1957)	La unidad atómica; ver sus límites motiva
2	El problema XOR	Por qué necesitamos no linealidad y profun
3	MLP: input → hidden → output	La arquitectura mínima útil.
4	Activación no lineal	Sin activación, N capas = 1 capa lineal.
5	Backpropagation (Rumelhart, Hinton & Willi	La regla de la cadena aplicada a un grafo.
6	Autograd / autodiff	Lo que hace que TF/PyTorch/JAX te liberen

Definiciones y características

- Perceptrón: $y = \text{step}(\Sigma\, w_i \cdot x_i + b)$. Regla de aprendizaje: $w_i \leftarrow w_i + \eta \cdot (y_{true} - y_{pred}) \cdot x_i$. Converge si los datos son linealmente separables.
- MLP (Multi-Layer Perceptron): red feedforward fully-connected con ≥ 1 capa oculta y activaciones no lineales (sigmoid, tanh, ReLU).
- Forward pass: $a^l = \sigma(W^l \cdot a^{l-1} + b^l)$. Se compute layer-by-layer hasta la salida.
- Loss function: L(y_pred, y_true) — MSE para regresión, cross-entropy para clasificación.
- Backward pass: aplica la regla de la cadena para calcular $\partial L/\partial W^l$ propagando el gradiente de la salida hacia la entrada.
- Autograd: TF (GradientTape), PyTorch (backward()), JAX (grad) — construyen el grafo y aplican backprop automáticamente.
- Universal Approximation Theorem (Cybenko 1989, Hornik 1991): un MLP con una sola capa oculta y "suficientes" neuronas puede aproximar cualquier función continua. En la práctica, profundo es más eficiente que ancho.

Dataset / recursos

- sklearn.datasets.make_moons(n_samples=500, noise=0.2) — clásico para mostrar no linealidad.
- XOR como dataset de 4 puntos (ejercicio motivacional).
- Librerías: tensorflow, keras (incluido), numpy, matplotlib.

Ejercicios

1. Perceptrón a mano: implementá un perceptrón en numpy y entrenalo en AND/OR (separables). Después probá con XOR; mostrá que no converge.
2. MLP con Keras: model = keras.Sequential([Dense(8, activation='relu', input_shape=(2,)), Dense(1, activation='sigmoid')]). Entrenalo sobre XOR; debería llegar a accuracy 1.0.
3. Visualización decision boundary: con make_moons, entrená un MLP [16, 8] y graficá la frontera con un meshgrid.
4. Backprop a mano: para un MLP con 1 input, 1 hidden (2 neuronas), 1 output, con MSE, calculá ∂L/∂w para una muestra y comparalo con tf.GradientTape.
5. ¿Y sin activación no lineal?: cambiá las activaciones a linear en el MLP de XOR y mostrá que ya no puede aprenderlo.

Homework verificable

Notebook que:

1. Carga make_moons(noise=0.2, random_state=42).
2. Entrena 2 modelos: regresión logística (Parte 1) vs MLP [16, 8] con ReLU.
3. Reporta accuracy en test para ambos.
4. Grafica las dos decision boundaries lado a lado.
5. Explica en 3 líneas por qué el MLP captura la curvatura y la regresión logística no.

Criterio de aceptación: el MLP debe lograr ≥ 0.95 accuracy y la frontera debe ser claramente curva; la regresión logística queda en ≈ 0.85 con frontera recta.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
MLP no aprende XOR aunque la arquitectura	Probablemente sin activación no lineal, o
Loss nan desde la primera época	Inicialización mala + activación + LR. Fix
Modelo sigmoid de salida con loss='mse'	Lento y mal calibrado. Fix: loss='binary_c
input_shape mal especificado: (2,) vs 2	Tiene que ser tupla. Fix: Dense(8, input_s
Modelo memoriza train y mal en test con XO	XOR son 4 puntos — no hay nada para genera

Preguntas frecuentes

¿Una sola neurona equivale a regresión logística?

Sí, exactamente: sigmoid(w·x + b) con cross-entropy = logistic regression. Por eso un MLP "es regresión logística apilada con no linealidades en medio".

¿Por qué no se podía entrenar redes profundas hasta los 2000s?

Por tres problemas: vanishing gradients (sigmoid satura), datasets chicos, hardware limitado. Lo resolvieron ReLU (2010), GPUs y ImageNet (2012). Lo verás en Clase 096.

¿Cuántas capas y neuronas pongo?

Empezá chico: 1-2 capas ocultas, 16-128 neuronas. Si underfittea, ensanchá o profundizá. Si overfittea, achicá o regularizá. La arquitectura se busca empíricamente con Keras Tuner (Clase 095) u Optuna.

¿Qué es el "Universal Approximation Theorem" en la práctica?

Que existe un MLP que aproxima tu función. No dice nada sobre cuán difícil es encontrarlo (el problema de optimización). Por eso "1 capa muy ancha" en teoría alcanza, pero "10 capas más angostas" en práctica

converge mucho mejor.

¿Forward + backward = una época?

No. Una iteración (= un batch) es 1 forward + 1 backward + 1 update. Una época es haber visto todo el dataset una vez (= n_samples / batch_size iteraciones).

Referencias

- Géron, cap. 10 — Introduction to Artificial Neural Networks with Keras.
- Rumelhart, Hinton & Williams (1986), Learning representations by back-propagating errors, Nature — paper original de backprop.
- Goodfellow, Bengio & Courville (2016), Deep Learning, cap. 6 (online: <https://www.deeplearningbook.org/>).
- 3Blue1Brown — serie "But what is a neural network?" (4 videos, intuición visual de backprop).
- Keras docs — Sequential model.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 101 — Clase 101 — Regresión y clasificación con MLP

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 10 § Building an Image Classifier y § Building a Regression MLP. Duración estimada: 75 min.*

Objetivo

Saber construir y entrenar un MLP para los tres tipos de problemas tabulares estándar — regresión, clasificación binaria y clasificación multiclase — eligiendo correctamente la activación de salida y la loss para cada caso. Hacer un train/val/test split adecuado y leer las curvas de entrenamiento.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Mapear problema → activación de salida → loss: regresión → linear + MSE; binario → sigmoid + binary_crossentropy; multiclase → softmax + sparse_categorical_crossentropy.
- Hacer split train / validation / test con train_test_split y pasar validation_data a model.fit.
- Leer history.history['loss'] y ['val_loss'], identificar overfitting (val sube mientras train baja).
- Aplicar EarlyStopping y ModelCheckpoint callbacks como protección estándar.
- Diferenciar sparse_categorical_crossentropy (labels enteros) de categorical_crossentropy (labels one-hot).

Temas

- Mapeo problema → arquitectura de salida + loss.
- Activación de salida: linear, sigmoid, softmax.
- Train/val/test split — por qué hace falta los tres (val para selección, test para reporte final).
- Curvas de aprendizaje: lectura visual (subfitting / overfitting).
- Callbacks: EarlyStopping(patience=5, restore_best_weights=True), ModelCheckpoint.

- Normalización de inputs con Normalization() layer o StandardScaler.

Definiciones y características

- linear activation: sin transformación. Salida no acotada. Para regresión.
- sigmoid: $1/(1+e^x)$, sale en (0, 1). Una neurona = probabilidad binaria.
- softmax: $e^{x_i} / \Sigma e^{x_j}$. Convierte logits en distribución de probabilidad sobre K clases.
- MSE: mean((y - ŷ)²). Para regresión. Sensible a outliers (cuadrado).
- MAE: mean(|y - ŷ|). Robusto a outliers.
- Binary Cross-Entropy: -[y·log(p) + (1-y)·log(1-p)]. Para 2 clases.
- Sparse Categorical Cross-Entropy: como categorical pero con labels enteros ([0, 2, 1, ...]). Más eficiente y común.
- EarlyStopping: corta entrenamiento cuando val_loss deja de bajar por patience épocas.

Dataset / recursos

- Regresión: sklearn.datasets.fetch_california_housing().
- Clasificación binaria: sklearn.datasets.load_breast_cancer().
- Multiclase: keras.datasets.fashion_mnist.load_data() (10 clases).
- Librerías: tensorflow, keras, scikit-learn, matplotlib.

Ejercicios

1. MLP regresión: California Housing, MLP [64, 32] con Normalization(), salida linear, loss mse. Reportar MAE en test.
2. MLP binario: breast_cancer, MLP [32, 16], salida sigmoid, loss binary_crossentropy. Reportar accuracy y AUC.
3. MLP multiclase: Fashion-MNIST (aplastado a 784), MLP [256, 128], salida softmax(10), loss sparse_categorical_crossentropy. Reportar accuracy.
4. Curvas y overfitting: entrenar el modelo 3 por 50 épocas sin early stopping. Graficar loss y val_loss; identificar la época donde arranca overfitting.
5. EarlyStopping: repetir con EarlyStopping(patience=5, restore_best_weights=True). Verificar que cortó antes y los pesos guardados son los mejores.

Homework verificable

Fashion-MNIST end-to-end:

1. Cargar y normalizar (/ 255).
2. Split train (50 000) / val (10 000) / test (10 000).
3. MLP [300, 100], ReLU, salida softmax.
4. Compilar con optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'].
5. Entrenar con EarlyStopping(patience=5) y ModelCheckpoint('best.keras', save_best_only=True).
6. Reportar accuracy en test y matriz de confusión.

Criterio de aceptación: accuracy en test ≥ 0.87; matriz de confusión muestra que las prendas más confundidas son Shirt / T-shirt / Pullover.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
loss = nan desde la época 1	Inputs sin escalar + LR alta. Fix: Standar
softmax + binary_crossentropy	Mismatch. Fix: softmax va con (sparse_)cat
categorical_crossentropy con labels entero	Espera one-hot. Fix: usar sparse_categoric

accuracy = 0.10 en Fashion-MNIST sin entre	Predicción aleatoria. Fix: model.fit(...).
Reportar accuracy en val como "test final"	Validation está contaminada por la búsqued

Preguntas frecuentes

¿Cuántas neuronas por capa?

Primera capa entre n_features y 4·n_features; capas posteriores más angostas (pirámide invertida). Ej.: 784 → 256 → 128 → 10. Refinar con tuning (Clase 095).

¿Qué optimizador uso?

Adam con LR default (1e-3) es el caballito de batalla. Para producción, AdamW con LR ajustada (Clase 102).

¿Cuántas épocas?

Las que decidan tus callbacks. Setear epochs=100 + EarlyStopping(patience=10).

¿Hay que normalizar siempre?

Siempre para features con escalas distintas. Para imágenes basta / 255. Sin normalizar, el optimizador zigzaguea.

¿Por qué la última capa de Fashion-MNIST tiene 10 neuronas?

Una neurona por clase. Softmax convierte los 10 logits en probabilidades que suman 1. El argmax es la predicción.

Referencias

- Géron, cap. 10 — Building an Image Classifier Using the Sequential API y Building a Regression MLP.
- Keras docs — Training & evaluation.
- Keras docs — Callbacks API.
- Glorot & Bengio (2010), Understanding the difficulty of training deep feedforward neural networks.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 102 — Clase 102 — Keras Sequential API

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 10 § The Sequential API. Duración estimada: 60 min.*

Objetivo

Dominar la Sequential API de Keras — la forma más simple y declarativa de construir un modelo cuando es una pila lineal de capas. Saber cuándo NO alcanza (cualquier topología con ramas, skip connections, multi-input/multi-output → Functional API, clase 093).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Construir un modelo con keras.Sequential([...]) o model.add(...) incrementalmente.

- Inspeccionar la arquitectura con model.summary() (parámetros por capa, output shape, total).
- Calcular a mano el número de parámetros de una Dense(n) (= input_dim * n + n por el bias).
- Compilar (compile), entrenar (fit), evaluar (evaluate) y predecir (predict).
- Guardar y cargar con el formato moderno .keras (HDF5 legacy).

Temas

- Dos formas equivalentes: lista en el constructor vs .add() incremental.
- Input layer explícito vs input_shape en la primera capa.
- model.summary(): leer parámetros por capa + total trainable / non-trainable.
- compile(optimizer, loss, metrics).
- fit(X, y, epochs, batch_size, validation_split, callbacks, verbose).
- model.save('m.keras') (formato nativo Keras 3+) y keras.models.load_model('m.keras').

Definiciones y características

- Sequential model: stack lineal — la salida de una capa es la entrada de la siguiente. No permite branching, skip connections, ni múltiples inputs/outputs.
- Parámetros trainable: pesos + bias que el optimizador actualiza. Dense(n_out) aplicada a entrada n_in: n_in * n_out + n_out params.
- Parámetros non-trainable: capas como BatchNormalization tienen estadísticas (running mean/var) que no se actualizan por backprop sino por moving average.
- .keras format: ZIP con config.json + metadata.json + pesos. Reemplaza al .h5 desde Keras 3.
- batch_size: cuántas muestras procesa antes de actualizar pesos. Default 32. Más grande = más estable pero menos updates por época.

Dataset / recursos

- Reutilizar Fashion-MNIST de la clase anterior.
- Librerías: tensorflow, keras (≥ 3.0).

Ejercicios

1. Dos sintaxis: construir el mismo modelo dos veces — una con Sequential([...]) y otra con model = Sequential(); model.add(...). Verificar que summary() produce el mismo resultado.
2. Conteo de parámetros: para Sequential([Dense(128, input_shape=(784,)), Dense(64), Dense(10)]), calcular a mano los parámetros y verificar contra model.summary().
3. Guardado/carga: entrenar 5 épocas, model.save('m.keras'), recargar con load_model, verificar que predict da idéntico.
4. Predict en batch vs individual: predecir 1 sola muestra (¿cómo cambia la shape?) vs predecir 100. Cuidado con la dimensión batch.
5. Verbose: probá fit(..., verbose=0), verbose=1 (barra), verbose=2 (1 línea por época). Útil para notebooks vs scripts.

Homework verificable

Entrenar tres arquitecturas distintas sobre Fashion-MNIST y comparar:

1. Tiny: Dense(64) → Dense(10).
2. Medium: Dense(256) → Dense(128) → Dense(10).
3. Wide: Dense(1024) → Dense(10).

Reportar: # parámetros, accuracy en test, tiempo de entrenamiento.

Criterio de aceptación: el medium debe ganar en accuracy/tiempo balanceado; el wide tiene más parámetros

que el medium pero NO necesariamente mejor accuracy (lección: profundo > ancho).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ValueError: Input shape mismatch al cargar	El modelo guardado tiene un input shape di
model.summary() muestra ? en output shape	No definiste el input shape de la primera
Olvido compile() y fit() falla	Sin compile, no hay optimizer/loss. Fix: m
Modelo guardado en .h5 y se quiere cargar	.h5 está deprecado. Fix: model.save('m.ker
model.add() después de compile() y fit() y	No se puede modificar la arquitectura post

Preguntas frecuentes

¿Cuándo Sequential no alcanza?

Cuando hay branching (skip connections de ResNet), multiple inputs (modelos de fusión), multiple outputs (multi-task), o capas compartidas. Para todo eso → Functional API (clase 093).

¿Es más lento Sequential que Functional?

No, son la misma cosa por dentro. La diferencia es solo sintáctica.

¿Por qué Input separado y no input_shape=?

Equivalentes funcionalmente. Input(shape=(...)) es más explícito y es lo único que funciona en Functional API; mantenerlo es buena costumbre.

¿batch_size=32 siempre?

Depende. Imágenes: 32–128. Texto/NLP: a menudo 8–16 por memoria. Tabular grande: 256–1024. Probá; el batch_size afecta no solo velocidad sino también la generalización (batches grandes tienden a converger a minima "más planos" pero peor generalizables).

¿validation_split=0.2 o validation_data=(X_val, y_val)?

validation_split=0.2 toma el último 20 % del array; útil si los datos están bien mezclados. validation_data=(...) es explícito y obligatorio si los datos están ordenados.

Referencias

- Géron, cap. 10 — The Sequential API.
- Keras — The Sequential model.
- Keras 3 release notes.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 103 — 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.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 104 — Clase 104 — Callbacks, TensorBoard, guardar/restaurar modelos

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 10 § Using Callbacks y § Using TensorBoard for Visualization. Duración estimada: 60 min.*

Objetivo

Inyectar lógica al loop de entrenamiento sin modificarlo, mediante callbacks (EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, custom). Visualizar el progreso del training en TensorBoard (loss, métricas, histogramas de pesos, embeddings). Saber guardar y restaurar correctamente — arquitectura + pesos + estado del optimizador.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar los 4 callbacks más usados: EarlyStopping, ModelCheckpoint, ReduceLROnPlateau, TensorBoard.
- Configurar TensorBoard: keras.callbacks.TensorBoard(log_dir='./logs') y abrirlo con tensorboard --logdir=./logs.
- Escribir un callback custom heredando de keras.callbacks.Callback con hooks como on_epoch_end.
- Distinguir guardado completo (model.save('m.keras')) vs solo pesos (model.save_weights('w.weights.h5')).
- Restaurar y continuar entrenamiento desde checkpoint sin pérdida.

Temas

- Callbacks: hooks (on_train_begin, on_epoch_end, on_batch_end, ...).
- EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True).
- ModelCheckpoint(filepath, save_best_only=True, monitor='val_accuracy', mode='max').
- ReduceLROnPlateau(factor=0.5, patience=5) — bajar LR cuando se estanca.
- TensorBoard: scalars, histograms, distributions, images, projector (embeddings).
- Custom callbacks: class MyCallback(keras.callbacks.Callback): def on_epoch_end(self, epoch, logs): ....

Definiciones y características

- Callback: objeto con métodos hook que Keras invoca en momentos específicos del training. Permite logging, early stopping, LR scheduling, etc.
- EarlyStopping: monitorea una métrica y corta cuando deja de mejorar. restore_best_weights=True revierte a los mejores pesos vistos.
- ModelCheckpoint: guarda el modelo (o solo pesos) cuando una métrica mejora.
- ReduceLROnPlateau: reduce el LR multiplicándolo por factor cuando la métrica se estanca por patience épocas.
- TensorBoard: la herramienta de visualización oficial de TF. Sirve via tensorboard --logdir=....
- logs dict: contiene loss, val_loss, métricas, etc. Disponible en hooks on_epoch_end.

Dataset / recursos

- Fashion-MNIST o cualquier modelo entrenable de clases anteriores.
- Librerías: tensorflow, keras, tensorboard (incluido).

Ejercicios

1. EarlyStopping + Checkpoint: entrenar Fashion-MNIST con ambos callbacks. Verificar que cortó cuando val_loss se estancó y que 'best.keras' contiene los mejores pesos.
2. TensorBoard: agregar TensorBoard(log_dir=f'./logs/run-{time}'), entrenar 10 épocas. Lanzar tensorboard --logdir=./logs y revisar scalars + histograms.
3. ReduceLROnPlateau: configurar factor=0.5, patience=3, min_lr=1e-6. Graficar el LR a lo largo de las épocas (usar el logs del callback).
4. Custom callback: escribir uno que loggee a un CSV el (epoch, loss, val_loss, lr_actual) para análisis offline.
5. Restaurar y continuar: entrenar 10 épocas, guardar, recargar y continuar 5 épocas más. Verificar que el optimizer state (momentum de Adam) se preservó.

Homework verificable

Entrenamiento "production-ready" sobre Fashion-MNIST:

1. MLP [300, 100].
2. Callbacks: EarlyStopping(patience=10), ModelCheckpoint('best.keras', save_best_only=True), ReduceLROnPlateau(patience=3), TensorBoard('./logs/').
3. Entrenar epochs=100 (sabiendo que EarlyStopping cortará antes).
4. Recargar best.keras y evaluar en test.
5. Capturar un screenshot del TensorBoard mostrando loss y val_loss a lo largo del training.

Criterio de aceptación: el modelo final restaurado debe ser el del epoch con menor val_loss; accuracy en test ≥ 0.87.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
EarlyStopping corta en la primera época co	Default es patience=0. Fix: patience=5-10
ModelCheckpoint guarda en cada época con s	Llena el disco. Fix: save_best_only=True.
model.save_weights('w.h5') y al cargar dic	Hay que reconstruir la arquitectura antes
TensorBoard no muestra nada en localhost:6	El log_dir está mal o aún no se escribió n
ReduceLROnPlateau y LearningRateScheduler	Conflicto, ambos cambian el LR. Fix: elegí

Preguntas frecuentes

¿save_best_only=True con qué métrica?

monitor='val_loss' por default (modo min). Si monitoreás accuracy, agregá mode='max'.

¿Puedo usar varios ModelCheckpoint?

Sí. Útil para guardar el mejor por val_loss y el mejor por val_accuracy en dos archivos.

¿TensorBoard funciona en Colab?

Sí: %load_ext tensorboard + %tensorboard --logdir=./logs. Funciona inline.

¿Custom callback necesita herencia explícita?

Sí: class MyCB(keras.callbacks.Callback). La base provee self.model y manejo de logs.

¿Format .keras vs .h5?

.keras (Keras 3+) es zip transparente, futuro-proof. .h5 aún funciona pero deprecado para nuevos proyectos.

Referencias

- Géron, cap. 10 — Using Callbacks y Using TensorBoard.
- Keras callbacks.
- TensorBoard quickstart.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 105 — 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.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 106 — 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.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 107 — Clase 107 — Vanishing/exploding gradients

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 11 § The Vanishing/Exploding Gradients Problems. Duración estimada: 70 min.*

Objetivo

Entender el problema central que estancó el Deep Learning hasta 2010: cuando un gradiente atraviesa muchas capas, se desvanece (sigmoid/tanh saturadas → multiplicas números < 1 muchas veces → ≈ 0) o explota (pesos grandes → > 1 muchas veces → ∞). Identificar los 4 culpables (activación, inicialización, profundidad, LR) y conocer las soluciones que destrabaron el campo: Glorot/He init (097), ReLU y variantes (098), BatchNorm (099), Gradient clipping (100).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diagnosticar vanishing gradient: gradients en capas tempranas con norma ~ 1e-8.
- Diagnosticar exploding: loss = nan o gradients con norma ~ 1e+10.
- Inspeccionar gradientes con tf.GradientTape + tf.norm.
- Mapear cada solución al problema: init para arrancar bien, ReLU para no saturar, BN para estabilizar, clipping para evitar explosión.
- Explicar por qué sigmoid en MLPs profundos no escala (derivada máx. 0.25 → gradiente decae rápido).

Temas

- Backprop como producto de derivadas a lo largo de capas.
- Sigmoid: σ(x) · (1 - σ(x)) máxima 0.25 en x=0; satura a 0 en colas.
- Tanh: derivada máx. 1, pero también satura.
- ReLU: derivada 0 o 1 — no decae al multiplicar, pero genera "dying ReLU".
- Inicialización mala: pesos N(0, 1) → activaciones explotan; pesos N(0, 0.01) → vanishing.
- BatchNorm: normaliza dentro del forward pass para que cada capa reciba inputs con varianza controlada.

Definiciones y características

- Vanishing gradient: las derivadas se acercan a 0 al propagar hacia atrás → capas tempranas no aprenden.
- Exploding gradient: las derivadas crecen sin límite → pesos divergen, loss = nan.
- Saturación: una activación está saturada cuando su derivada ≈ 0 (sigmoid en |x| > 5).
- Dying ReLU: una neurona ReLU queda con salida siempre 0 → gradiente 0 → no se actualiza más. Solución: Leaky ReLU, ELU, init He.
- Gradient norm: ||∂L/∂W||. Indicador visualizable durante el entrenamiento.

Dataset / recursos

- Fashion-MNIST.
- Librerías: tensorflow, keras, matplotlib.

Ejercicios

1. Diagnóstico vanishing: entrenar un MLP de 10 capas con sigmoid activations e init default. Inspeccionar gradientes de la primera capa vs la última. Verificar que los de la primera son órdenes de magnitud más chicos.
2. Mismo experimento con ReLU: comparar gradients. Mejor pero aún heterogéneo.
3. Exploding: forzar init RandomNormal(stddev=5). Observar loss = nan en pocas iteraciones.
4. Norma del gradiente por capa: con tf.GradientTape, calcular tf.norm(g) para cada peso y graficar a lo largo del entrenamiento.
5. Solución sencilla: cambiar a He init + ReLU + BatchNorm y mostrar que el problema desaparece (anticipa las siguientes 4 clases).

Homework verificable

Construir un MLP de 20 capas y reportar entrenamiento bajo 4 condiciones:

1. Sigmoid + Glorot init.
2. ReLU + Glorot init.
3. ReLU + He init.

4. ReLU + He init + BatchNorm.

Para cada uno: graficar loss durante 20 épocas + reportar val_accuracy final.

Criterio de aceptación: (1) prácticamente no aprende; (2) aprende lento; (3) aprende razonable; (4) aprende rápido y mejor. La diferencia visual debe ser dramática.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
loss = nan	Exploding. Fix: bajar LR, agregar gradient
loss se queda en valor constante alto	Vanishing. Fix: ReLU + He init.
Accuracy en train baja, val baja	El modelo no aprende. Diagnóstico: imprimi
50 % de las neuronas tienen salida 0 todo	Dying ReLU. Fix: Leaky ReLU o ELU.
Después de unas épocas la loss salta a nan	Exploding tardío. Fix: gradient clipping (

Preguntas frecuentes

¿Por qué sigmoid no funciona en MLPs profundos pero sí en la última capa?

Porque la última capa tiene solo 1 multiplicación; las capas profundas multiplican N veces. $(0.25)^{10} \approx 10^{6} \rightarrow$ desaparece. En la última capa, sigmoid sí sirve para probabilidades.

¿Cuánto es "profundo"?

A partir de 5-10 capas Dense (más con BN). En CNNs, con BN o residuals se llega a 100+. En Transformers, decenas pueden no usar residuals + LayerNorm = vanishing.

¿BatchNorm "soluciona" todo?

Resuelve el vanishing/exploding casi completamente al normalizar las activaciones. Pero tiene problemas propios (batch chico, dependencia entre muestras) — clase 099 entra en detalle.

¿GELU/Swish ayudan?

Sí. Son activaciones más suaves que ReLU (sin "muerte"), usadas en transformers modernos (BERT, GPT). Clase 098.

¿Esto sigue siendo relevante con LLMs?

Sí — los LLMs usan combinación de LayerNorm + residual + GELU + init cuidadoso (Xavier/Glorot custom) precisamente para esto. Sin estas piezas, no se podrían entrenar.

Referencias

- Géron, cap. 11 — Training Deep Neural Networks.
- Hochreiter (1991), thesis — primera descripción del vanishing gradient.
- Bengio, Simard & Frasconi (1994), Learning Long-Term Dependencies with Gradient Descent is Difficult.
- Glorot & Bengio (2010), Understanding the difficulty of training deep feedforward neural networks, AISTATS.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.

- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 108 — Clase 108 — Inicialización (Glorot, He)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 11 § Glorot and He Initialization. Duración estimada: 55 min.*

Objetivo

Saber inicializar los pesos de cada capa para que la varianza de las activaciones y de los gradientes se mantenga estable a lo largo del forward y backward pass. Diferenciar Glorot (Xavier) —para sigmoid/tanh— de He (Kaiming) —para ReLU y variantes—. Saber cuál usa Keras por default y cuándo cambiarlo.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar la idea: Var(W) ≈ 1 / fan_in (o promedio fan_in/fan_out) para preservar varianza.
- Aplicar kernel_initializer='glorot_uniform' (default Keras), 'he_normal', 'he_uniform'.
- Calcular a mano los límites de la distribución para Glorot uniform: ±√(6/(fan_in+fan_out)).
- Reconocer que la combinación correcta es He init + ReLU, Glorot + tanh/sigmoid.
- Inspeccionar el efecto visualmente: histogramas de activaciones por capa.

Temas

- ¿Por qué importa la varianza? Productos de N capas amplifican o atenúan exponencialmente.
- Glorot (2010): Var(W) = 2/(fan_in + fan_out). Asume activación lineal/simétrica.
- He (2015): Var(W) = 2/fan_in. Compensa que ReLU "mata" la mitad de las salidas.
- Distribuciones: uniform o normal. Equivalentes prácticamente.
- LeCun init: Var(W) = 1/fan_in. Para SELU.

Definiciones y características

- fan_in: número de entradas a la capa (= units de la capa anterior).
- fan_out: número de salidas (= units de la capa).
- Glorot uniform: U(-√(6/(fan_in+fan_out)), +√(6/(fan_in+fan_out))). Default de Keras Dense.
- He normal: N(0, √(2/fan_in)). Recomendado para ReLU, Leaky ReLU, ELU.
- LeCun normal: N(0, √(1/fan_in)). Para SELU (self-normalizing networks).
- Inicialización por capa: Dense(64, kernel_initializer='he_normal', bias_initializer='zeros').

Dataset / recursos

- Fashion-MNIST.
- Librerías: tensorflow, keras, matplotlib.

Ejercicios

1. Inspección de defaults: para Dense(128, input_shape=(784,)), imprimir model.layers[0].kernel.numpy(). Calcular la varianza empírica y compararla con la teórica de Glorot.
2. Comparación: entrenar MLP [512, 256, 128, 64, 10] con ReLU. Probar 3 inits: Glorot, He, RandomNormal(stddev=0.01). Graficar val_loss en las 3.
3. Histogramas de activaciones: para cada capa del modelo bien inicializado, plot del histograma de salidas para un batch. Verificar que la varianza se mantiene similar entre capas.

4. He init + Tanh: probar la combinación incorrecta (He con tanh). Comparar contra Glorot + tanh. Verificar que importa.
5. Reset y reproducibilidad: con tf.random.set_seed(42) + np.random.seed(42), entrenar 2 veces y verificar que da idéntico. Sin seed, varía.

Homework verificable

Sobre MLP [300, 200, 100, 50, 10] con ReLU sobre Fashion-MNIST:

1. Entrenar con 3 inits: default Glorot, He uniform, He normal.
2. Reportar val_accuracy tras 20 épocas para cada uno.
3. Para el mejor (He init), inspeccionar la norma de cada kernel antes y después del entrenamiento.

Criterio de aceptación: He init debe igualar o superar Glorot por ≥ 0.5 pp; la norma de los kernels post-entrenamiento debe ser similar entre capas (no orders of magnitude apart).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Modelo arranca lento con ReLU	Estás usando Glorot por default. Fix: kern
RandomNormal(stddev=0.01) por costumbre de	Vanishing inmediato. Fix: usar Glorot/He s
Reset de pesos no funciona con model.set_w	Hay que guardar initial = [w.numpy() for w
Bias init glorot_uniform por error	Bias debe arrancar en zeros (default). Ini
Inicializar igual una capa convolucional q	fan_in para Conv es kernel_h × kernel_w ×

Preguntas frecuentes

¿Glorot uniform o normal?

Para Glorot, casi indistinguible en práctica. Keras default es glorot_uniform. Para He, he_normal es ligeramente preferido pero las diferencias son ínfimas.

¿LeCun init cuándo?

Solo con SELU (activación que se auto-normaliza). Esto fue un experimento de 2017 (Klambauer et al.) que perdió tracción frente a BatchNorm + ReLU/GELU.

¿BatchNorm hace innecesaria la inicialización?

Casi. BN normaliza el output dentro del forward → init importa menos. Pero un mal init aún hace que las primeras épocas sean inestables.

¿Init para Transformers?

Combinación cuidadosa: linears con truncated_normal(stddev=0.02) (GPT/BERT-style), embeddings con normales escaladas. Lo verás en clase 126.

¿Por qué fan_in + fan_out en Glorot?

Es el promedio armónico de "preservar varianza en forward (1/fan_in)" y "preservar varianza en backward (1/fan_out)". Compromiso óptimo bajo activación lineal.

Referencias

- Géron, cap. 11 — Glorot and He Initialization.
- Glorot & Bengio (2010), Understanding the difficulty of training deep feedforward neural networks,

AISTATS.
- He, Zhang, Ren & Sun (2015), Delving Deep into Rectifiers, ICCV — paper original de He init.
- Keras initializers.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 109 — Clase 109 — Activaciones: ReLU, ELU, GELU, Swish, Mish

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 11 § Better Activation Functions. Duración estimada: 60 min.*

Objetivo

Conocer la familia de activaciones modernas — desde ReLU (Krizhevsky et al. 2012) hasta GELU (BERT, GPT) y Swish/SiLU (EfficientNet) — entendiendo qué problema resuelve cada una y por qué los Transformers modernos usan GELU y no ReLU. Saber elegir según arquitectura.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Definir matemáticamente las 5 activaciones: ReLU, Leaky ReLU, ELU, GELU, Swish (SiLU), Mish.
- Identificar dying ReLU y aplicar Leaky ReLU / ELU como mitigación.
- Reconocer que GELU es la activación default en Transformers (BERT, GPT, ViT) y Swish/SiLU en EfficientNet, modelos modernos de visión.
- Aplicar cada activación en Keras: Dense(64, activation='relu' | 'gelu' | 'swish' | 'elu' | LeakyReLU()).
- Saber que el costo computacional de GELU/Swish es mayor (sigmoid/erf internos) pero el beneficio supera en arquitecturas profundas.

Temas

- ReLU: max(0, x). Rápida, simple, default histórico. Dying ReLU.
- Leaky ReLU: max(αx, x) con α≈0.01. Sin dying.
- ELU (Clevert et al. 2015): x si x>0, α(e^x-1) si x<0. Suave, sin dying, pero más cara.
- GELU (Hendrycks & Gimpel 2016): x · Φ(x) (Φ = CDF gaussiana). Suave, no monótona. Default en Transformers.
- Swish / SiLU (Ramachandran et al. 2017): x · sigmoid(x). Encontrada por NAS. Casi idéntica a GELU.
- Mish (Misra 2019): x · tanh(softplus(x)). Marginalmente mejor en algunos benchmarks.

Definiciones y características

- Función de activación: no linealidad aplicada elementwise tras la transformación lineal de una capa.
- Saturación: cuando la derivada se acerca a 0 → vanishing.
- Monótona: f(x) creciente en todo el dominio. ReLU/ELU/Leaky lo son; GELU/Swish/Mish no estrictamente.
- Bounded: salida acotada. Sigmoid (0,1), tanh (-1,1), softmax. La mayoría modernas no son bounded.
- Smooth: derivable continuamente. ReLU no en x=0; las demás sí.

Dataset / recursos

- Fashion-MNIST.
- Librerías: tensorflow, keras.

Ejercicios

1. Plot de funciones: graficar las 6 activaciones en x [-3, 3].
2. Comparación empírica: entrenar MLP [256, 128, 64] con cada activación, mismo init He, mismo LR. Comparar val_accuracy tras 15 épocas.
3. Dying ReLU: con LR alto (0.1), entrenar con ReLU. Contar cuántas neuronas tienen mean(activation) = 0 al final.
4. Leaky ReLU al rescate: repetir con Leaky. Verificar que el % de neuronas muertas baja.
5. GELU vs ReLU en profundidad: armar un MLP de 12 capas. Comparar GELU vs ReLU. GELU suele ganar.

Homework verificable

Sobre Fashion-MNIST, MLP [300, 200, 100, 50, 10]:

1. Entrenar 4 modelos con: ReLU, Leaky ReLU(α=0.1), ELU, GELU.
2. Reportar val_accuracy y tiempo por época.
3. Para el mejor, inspeccionar % de neuronas "muertas" por capa.

Criterio de aceptación: GELU o ELU debe ganar en accuracy; Leaky/ELU/GELU deben tener < 5 % neuronas muertas; ReLU posiblemente más.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Cambio de ReLU a sigmoid en capa oculta pa	Vanishing immediato en redes profundas. Fi
Leaky ReLU con α=0.5	Casi lineal, pierde la no linealidad útil.
activation='swish' no reconocido	Keras antiguo lo llama 'swish', TF moderno
GELU en CPU es lento	Es ≈ 2× más caro que ReLU por la erf. En C
Mish en GPU sin implementación optimizada	TF puede no tener kernel cuda eficiente pa

Preguntas frecuentes

¿Qué activación uso por default en 2026?

- MLP/CNN clásica: ReLU o GELU.
- Transformers: GELU (es lo que usan BERT, GPT, ViT).
- EfficientNet/MobileNet: Swish (= SiLU).
- Modelos profundos sin BN: ELU o GELU (más estables).

Swish y SiLU son lo mismo?

Sí, mismo formato (x · sigmoid(x)). SiLU es el nombre PyTorch; Swish es el nombre de Google.

¿Por qué GELU en Transformers?

Históricamente: BERT (2018) la eligió porque parecía funcionar mejor empíricamente. Hoy es estándar más por inercia/compatibilidad que por una ventaja categórica. En benchmarks recientes, Swish es competitiva.

¿GELU exacta o aproximada?

tf.keras.activations.gelu(x, approximate=False) usa erf exacta. True usa la aproximación tanh, ≈10× más rápida en GPU. La diferencia numérica es < 0.001.

¿La salida del modelo también lleva GELU?

No. La activación final depende del problema (linear, sigmoid, softmax). GELU/ReLU son para capas ocultas.

Referencias

- Géron, cap. 11 — Better Activation Functions.
- Krizhevsky et al. (2012), AlexNet — ReLU al mainstream.
- Clevert et al. (2015), ELU, ICLR.
- Hendrycks & Gimpel (2016), Gaussian Error Linear Units (GELUs).
- Ramachandran et al. (2017), Searching for Activation Functions, ICLR — Swish.
- Misra (2019), Mish: A Self Regularized Non-Monotonic Activation Function.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 110 — Clase 110 — Batch Normalization, Layer Normalization

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 11 § Batch Normalization. Duración estimada: 75 min.*

Objetivo

Entender BatchNorm (Ioffe & Szegedy 2015) — la técnica que destrabó el entrenamiento de redes muy profundas estandarizando las activaciones en cada capa — y su variante LayerNorm (Ba, Kiros & Hinton 2016) — usada en Transformers y RNN porque no depende del batch. Saber dónde poner BN en la arquitectura, qué problemas tiene (batch chico, distribución entre train/inference) y cuándo preferir LN.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar BatchNormalization() antes o después de la activación (debate clásico — moderno: antes suele ser mejor para ReLU, después para GELU).
- Explicar qué hace BN en train (normaliza con stats del batch) vs inference (usa moving averages acumulados).
- Aplicar LayerNormalization() en RNN y Transformers; saber por qué allí BN falla.
- Reconocer las 3 variantes: BN, LN, GroupNorm (Wu & He 2018, para batch chico en visión).
- Diagnosticar el problema de "train-test mismatch" cuando el batch en inference es muy distinto al de train.

Temas

- BN: $y = \gamma \cdot (x - \mu_{batch})/\sigma_{batch} + \beta$. γ, β trainables.
- Beneficios: convergencia más rápida, regularización mild, permite LR más altos.
- BN en train vs inference: moving avg de μ, σ acumulados.
- LN: normaliza sobre los features de una sola muestra (no sobre el batch).
- GroupNorm: agrupa canales, normaliza dentro de cada grupo. Para batch chico (segmentación, detección).
- ¿Antes o después de la activación? Géron y la práctica moderna: antes funciona mejor con ReLU,

después con GELU/Swish.

Definiciones y características

- BatchNorm: normaliza cada feature usando la media y varianza del batch actual durante training.
- γ (scale), β (shift): parámetros learnables que permiten al modelo deshacer la normalización si necesita.
- Moving averages: durante training, mantiene EMAs de μ, σ. En inference usa esos valores fijos.
- LayerNorm: normaliza sobre features de una sola muestra, independiente del batch. Default en NLP/Transformers.
- GroupNorm: agrupa N canales y normaliza dentro de cada grupo. Funciona con batch_size=1.
- InstanceNorm: como BN pero por sample individual (estilo style transfer).

Dataset / recursos

- Fashion-MNIST + un modelo profundo ([512, 256, 128, 64, 10]).
- Librerías: tensorflow, keras.

Ejercicios

1. BN vs sin BN: entrenar el mismo MLP con y sin BN. Comparar curvas de val_loss y tiempo hasta llegar a accuracy 0.85.
2. ¿Antes o después de la activación?: probar las dos variantes (Dense → BN → ReLU vs Dense → ReLU → BN). Comparar.
3. Inference mode: entrenar con BN, cambiar a training=False, predecir un batch y comparar con training=True. Las predicciones cambian (correcto).
4. Batch chico: forzar batch_size=4 y entrenar con BN. Observar inestabilidad. Cambiar a LayerNormalization y verificar que se estabiliza.
5. LayerNorm en RNN: aplicar keras.layers.LSTM con recurrent_activation y un LayerNormalization previo. Útil como anticipo de Transformers.

Homework verificable

Sobre MLP [512, 256, 128, 64, 10] en Fashion-MNIST:

1. Entrenar 3 versiones: sin norm, con BN, con LN.
2. Reportar val_accuracy + épocas hasta val_loss < 0.4.
3. Entrenar BN con batch_size=128 y luego inferir batch por batch de tamaño 1. Verificar que el accuracy no se rompe (gracias a moving averages).

Criterio de aceptación: BN debe acelerar la convergencia ≥ 30 % (menos épocas para mismo loss); LN debe ser estable también pero típicamente algo peor en MLP feedforward.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
BN en batch_size=1 → entrenamiento inestab	μ del batch es la muestra misma → σ=0. Fix
Métricas en val muy distintas de train par	Las stats de BN en inference son distintas
Olvidar pasar training=True/False en custo	Por default es False → BN usa moving avg i
BN en una RNN	Las estadísticas cambian con la longitud d
Aplicar BN a la última capa antes de softm	Generalmente innecesario y a veces dañino

Preguntas frecuentes

¿BN reemplaza dropout?

Parcialmente. BN tiene efecto regularizador mild (el ruido del batch). En CNNs/MLPs profundos suele alcanzar; en Transformers se usa LN + dropout juntos.

¿LN o BN en Transformers?

LN siempre. BN falla porque (a) batch chico es común, (b) longitudes variables rompen las estadísticas.

¿Por qué LN no es default también en CNN?

Empíricamente BN gana en visión (los canales se benefician de stats compartidos). LN gana en NLP donde el orden temporal importa más.

¿fused=True qué hace?

BatchNormalization(fused=True) usa una implementación CUDA optimizada que fusiona ops. Default auto lo elige cuando puede. Velocidad ~ 20 % más rápido.

¿Cuánto cuesta BN en cómputo?

Casi gratis en GPU (ops elementwise + pocas reducciones). En CPU es notable pero negligible en práctica.

Referencias

- Géron, cap. 11 — Batch Normalization.
- Ioffe & Szegedy (2015), Batch Normalization, ICML.
- Ba, Kiros & Hinton (2016), Layer Normalization.
- Wu & He (2018), Group Normalization, ECCV.
- Santurkar et al. (2018), How Does Batch Normalization Help Optimization?, NeurIPS — explicación moderna.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 111 — Clase 111 — Gradient clipping

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 11 § Gradient Clipping + Pascanu, Mikolov & Bengio (2013). Duración estimada: 45 min.*

Objetivo

Aplicar gradient clipping —limitar la norma o el valor de los gradientes antes de actualizar pesos— como protección contra exploding gradients, especialmente crítico en RNN/LSTM (clase 120) y en entrenamiento de LLMs. Diferenciar clipnorm (preserva dirección) de clipvalue (clipea por elemento).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Configurar clipping en cualquier optimizer Keras: Adam(clipnorm=1.0) o Adam(clipvalue=0.5).
- Saber cuándo clipnorm es preferible (default moderno): preserva dirección del gradiente.
- Implementar clipping manual en custom training loop con tf.clip_by_global_norm.
- Detectar exploding monitoreando la norma del gradiente.

- Reconocer que en Transformers de LLM, clipnorm=1.0 es estándar.

Temas

- Exploding revisitado: ¿qué pasa cuando ||grad|| crece exponencialmente?
- clipnorm: si ||g|| > c, escalar g ← g · c/||g||. Preserva dirección.
- clipvalue: g_i ← clip(g_i, -c, +c) por elemento. Cambia dirección.
- Global norm vs per-variable: clip_by_global_norm mira el norm del tensor concatenado de todos los pesos.

Definiciones y características

- Gradient clipping: limitar el tamaño del gradiente antes de aplicarlo.
- clipnorm: norma euclídea L2 máxima permitida. Si excede, se reescala manteniendo dirección.
- clipvalue: máximo valor absoluto por elemento.
- Global norm: ||g|| calculado sobre todos los gradientes concatenados como un solo vector.
- tf.clip_by_global_norm(grads, clip_norm=1.0): API moderna para clipping en custom loops.

Dataset / recursos

- Fashion-MNIST + un MLP propenso a exploding (LR alto + sin BN).
- Librerías: tensorflow, keras.

Ejercicios

1. Forzar exploding: entrenar MLP con Adam(lr=10.0) sobre Fashion-MNIST. loss = nan rápido.
2. Clipping al rescate: repetir con Adam(lr=10.0, clipnorm=1.0). Verificar que no explota (aunque sigue malo el LR — clipping no es solución a LR mal calibrado, solo a explosión).
3. clipnorm vs clipvalue: comparar las dos con LR razonable. Para problemas estables son ~equivalentes; diferencias aparecen en patrones específicos.
4. Custom loop: implementar el paso con gradients = tape.gradient(loss, model.trainable_variables); gradients, _ = tf.clip_by_global_norm(gradients, 1.0); optimizer.apply_gradients(...).
5. Monitoreo: graficar ||grad|| por step. Verificar que no excede el clipnorm configurado.

Homework verificable

Reentrenar el modelo del ejercicio anterior con LR razonable + clipnorm=1.0 como práctica estándar:

1. MLP [300, 100] Fashion-MNIST.
2. Adam(learning_rate=1e-3, clipnorm=1.0).
3. Graficar la norma del gradiente en cada step (custom loop o callback).
4. Verificar que la curva está acotada a 1.0 cuando el modelo aún no convergió.

Criterio de aceptación: el modelo entrena estable (val_accuracy ≥ 0.87) y la norma del gradiente está acotada como esperado.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
clipnorm=0.01 y modelo no aprende	Clipping muy agresivo enmascara gradientes
Configuro clipvalue y los gradientes peque	Si el valor es chico y los gradientes son
Custom loop sin clipping → loss=nan ocasio	Sin clipping RNN explota. Fix: tf.clip_by_
Cliping pasa pero el problema es otro (LR	Clipping no arregla LR mal calibrado. Diag
clipnorm y clipvalue simultáneamente	Keras los aplica en cascada — comportamien

Preguntas frecuentes

¿clipnorm=1 o clipnorm=5?

1.0 para Transformers/LLMs (estándar). 5.0 para RNN/LSTM clásicos. Para MLPs/CNNs con BN, en general no hace falta clipping; un default de 1.0 no hace daño.

¿Clipping deteriora el modelo final?

Si el clipping nunca se activa (rara vez sobrepasás), no hace nada. Si se activa todo el tiempo, es una banda-aid sobre un problema más serio. Ideal: clipnorm por si las moscas pero no debería gatillarse seguido.

¿Cuándo monitorear la norma del gradiente?

Siempre que entrenes algo nuevo / no probado. Es el indicador más rápido de exploding/vanishing.

¿GradientTape global vs por variable?

Global norm (clip_by_global_norm) es lo correcto: trata el conjunto de pesos como un vector único. Per-variable distorsiona la dirección.

¿Por qué en LLMs es tan importante?

Transformers profundos + sequences largas → gradientes pueden ser muy heterogéneos. clipnorm=1.0 y Adam(beta1=0.9, beta2=0.95) son la receta default de modelos como GPT-3 entrenados desde cero.

Referencias

- Géron, cap. 11 — Gradient Clipping.
- Pascanu, Mikolov & Bengio (2013), On the difficulty of training recurrent neural networks, ICML.
- tf.clip_by_global_norm.
- Keras Optimizer base — clipnorm/clipvalue/global_clipnorm.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 112 — Clase 112 — Transfer learning, unsupervised pretraining

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 11 § Reusing Pretrained Layers. Duración estimada: 70 min.*

Objetivo

Aplicar transfer learning — el patrón dominante en producción cuando hay pocos datos: tomar un modelo preentrenado (ImageNet para visión, BERT/GPT para texto), reemplazar la cabeza, congelar las capas base, fine-tunear la cabeza, y opcionalmente descongelar gradualmente para una segunda fase con LR muy bajo. Conocer unsupervised pretraining como hermano histórico (autoencoders, contrastive learning).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Cargar un modelo preentrenado: keras.applications.MobileNetV3Small(weights='imagenet',

include_top=False).
- Congelar capas: base.trainable = False.
- Construir un modelo nuevo con la base + un head custom (GlobalAveragePooling2D + Dense).
- Fine-tunear en dos etapas: (a) solo head con LR normal, (b) toda la red con LR 10× más bajo.
- Reconocer cuándo NO usar transfer (dataset muy distinto al de origen).

Temas

- Por qué funciona: las capas tempranas aprenden features generales (bordes, texturas); las tardías son task-specific.
- Pipeline estándar: load → freeze → new head → fit → unfreeze → fit con LR bajo.
- LR diferencial: capas tempranas más bajo (1e-5), capas tardías más alto (1e-3).
- Unsupervised pretraining: autoencoders (clase 130), contrastive (SimCLR, MoCo, CLIP).
- Self-supervised: cómo BERT/GPT se pre-entrenan sin labels (masked LM / autoregressive).

Definiciones y características

- Transfer learning: reusar pesos preentrenados en una tarea como punto de partida para otra.
- Feature extraction: usar el modelo base como extractor fijo (trainable=False) y entrenar solo la cabeza.
- Fine-tuning: descongelar (parte o todo) el modelo base y continuar entrenando con LR bajo.
- Catastrophic forgetting: si descongelás y usás LR alto, el modelo "olvida" lo aprendido. Por eso LR bajo en fine-tuning.
- Frozen / Trainable: layer.trainable = False excluye los pesos del backprop.
- Self-supervised: pre-entrenar con un objetivo derivado de los propios datos (predecir palabra masked, contrastar augmentaciones).

Dataset / recursos

- Visión: dataset chico de 2-5 clases (perros/gatos, flores). tf.keras.utils.image_dataset_from_directory.
- Modelo base: MobileNetV3Small o EfficientNetB0 (más liviano que ResNet50).
- Librerías: tensorflow, keras, keras.applications.

Ejercicios

1. Carga: base = MobileNetV3Small(weights='imagenet', include_top=False, input_shape=(224,224,3)); base.trainable = False.
2. Modelo completo: model = Sequential([base, GlobalAveragePooling2D(), Dropout(0.2), Dense(num_classes, activation='softmax')]).
3. Etapa 1: compilar con Adam(1e-3) y entrenar 10 épocas. Reportar accuracy.
4. Etapa 2: base.trainable = True y recompilar con Adam(1e-5). Entrenar 10 épocas más. Verificar mejora.
5. Sin transfer: entrenar la misma arquitectura desde cero (weights=None). Comparar cuántas épocas necesita para igualar accuracy de transfer (típicamente: nunca lo iguala con dataset chico).

Homework verificable

Clasificación de un dataset propio de imágenes (≥ 3 clases, ≥ 100 imágenes por clase):

1. Pipeline con image_dataset_from_directory + augmentation (RandomFlip, RandomRotation).
2. Transfer learning con EfficientNetB0.
3. Reportar accuracy de las dos etapas (frozen y unfreeze).
4. Comparar contra entrenar desde cero.

Criterio de aceptación: la etapa 2 (unfreeze + LR bajo) debe mejorar la etapa 1 por ≥ 2 pp; el modelo desde cero queda al menos 10 pp por debajo.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Olvido base.trainable = False y la etapa 1	Está entrenando millones de params. Fix: c
Recompilar entre etapas pero trainable cam	El cambio de trainable requiere recompilar
Imágenes en [0, 255] cuando el modelo espe	Pre-procesar con keras.applications.<model
BN en training=True se actualiza en fine-t	Sin cuidado, las moving averages se siguen
Catastrophic forgetting	LR demasiado alto al descongelar. Fix: 10-

Preguntas frecuentes

¿Cuántas imágenes mínimas para transfer learning?

Para fine-tuning: 100-500 por clase. Para feature extraction (solo head): incluso 20-50 por clase puede funcionar.

¿Qué modelo base elijo?

Empezá con MobileNetV3Small o EfficientNetB0 (chicos, rápidos). Si necesitás más capacidad y tenés GPU, EfficientNetB3-B5 o ConvNeXt.

¿Cuántas capas descongelo?

Empezá con todo descongelado + LR bajo. Si overfittea, congelar las primeras N capas (las más genéricas).

¿Transfer funciona para datos médicos / satelitales?

Sí, pero menos: las features de ImageNet son menos relevantes. Aún así, suele superar entrenar desde cero. Para datasets muy específicos, considerar pretraining con MAE o SimCLR sobre los datos propios.

¿Y para texto?

Misma idea: cargar bert-base o distilbert desde Hugging Face (clase 127), agregar head, fine-tunear. Estándar industrial.

Referencias

- Géron, cap. 11 — Reusing Pretrained Layers.
- Keras Applications — catálogo de modelos preentrenados.
- Pan & Yang (2010), A Survey on Transfer Learning, IEEE TKDE.
- Chen et al. (2020), A Simple Framework for Contrastive Learning of Visual Representations (SimCLR), ICML.
- He et al. (2022), Masked Autoencoders Are Scalable Vision Learners (MAE), CVPR.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 113 — Clase 113 — Optimizadores: Momentum, Nesterov, AdaGrad, RMSProp, Adam, AdamW (+ Lion, Sophia)

Parte: 2 — Deep Learning · Fuente: Géron, cap. 11 § Faster Optimizers + papers Lion (Chen et al. 2023), Sophia (Liu et al. 2023). Duración estimada: 80 min.

Objetivo

Conocer la evolución de los optimizadores —SGD → Momentum → Nesterov → AdaGrad → RMSProp → Adam → AdamW— entendiendo qué problema resuelve cada uno. Aplicar los optimizadores 2023+ (Lion, Sophia) que están reemplazando a Adam en LLMs grandes por mejor performance y memoria. Saber elegir según contexto (Adam para casi todo, SGD+momentum para visión clásica, Lion para LLMs).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar la fórmula de cada uno: SGD (w ← w - η·g), Momentum (acumulación), Nesterov (lookahead), Adam (mom 1er + 2do orden).
- Diferenciar Adam vs AdamW — la corrección de weight decay que Loshchilov & Hutter (2019) demostraron esencial.
- Usar Lion (tf.keras.optimizers.Lion en Keras 3+) con LR 3-10× más bajo que Adam.
- Reconocer cuándo SGD+Momentum supera a Adam: visión clásica con datasets grandes (ImageNet), donde el modelo final generaliza mejor.
- Inspeccionar y entender los hiperparámetros beta_1, beta_2, epsilon, weight_decay.

Temas

- SGD vanilla y por qué es lento en cañones (zigzaguea).
- Momentum (Polyak 1964): acelera en direcciones consistentes.
- Nesterov (1983): "miro hacia adelante" antes de calcular gradiente.
- AdaGrad: tasa adaptativa por parámetro; bueno para sparse data, malo para LR que decae a 0.
- RMSProp (Hinton, sin publicar): suaviza AdaGrad con EMA.
- Adam (Kingma & Ba 2014): Momentum + RMSProp = caballito industrial.
- AdamW (Loshchilov & Hutter 2019): weight decay separado del gradiente.
- Complemento moderno: Lion (Chen et al. 2023, signo en lugar de gradient adaptive), Sophia (Liu et al. 2023, Hessian aproximada).

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 102b — Optimizadores modernos: Lion, Sophia, Schedule-Free

Definiciones y características

- SGD: w ← w - η · g. La base.
- Momentum: v ← β·v + g; w ← w - η·v. β típicamente 0.9.
- Nesterov: como momentum, pero calcula el gradiente en el punto adelantado.
- AdaGrad: s += g²; w ← w - η·g/√s. LR efectivo decrece para parámetros frecuentes.
- RMSProp: como AdaGrad pero con EMA: s ← β·s + (1-β)·g².
- Adam: combina momentum (1er momento) + RMSProp (2do momento) + bias correction.
- AdamW: aplica weight decay como una operación separada del gradiente (decoupled), no como L2.
- Lion: gradient sign + momentum. Menos memoria.
- beta_1: decay del primer momento (default 0.9).
- beta_2: decay del segundo momento (default 0.999 Adam / 0.99 Lion).

- epsilon: estabilidad numérica del √. Default 1e-7. En LLMs a veces 1e-8 / 1e-5.
- weight_decay: en AdamW, el coeficiente de "tirar pesos hacia 0" — regularización L2 implícita.

Dataset / recursos

- Fashion-MNIST + un modelo razonable.
- Librerías: tensorflow, keras (Lion incluido en Keras 3+).

Ejercicios

1. Comparar 5 optimizadores: SGD(0.01), SGD+Momentum(0.9), Adam(1e-3), AdamW(1e-3, wd=1e-2), Lion(1e-4, wd=0.1). Mismo modelo, mismo dataset, 20 épocas. Graficar val_loss.
2. Tuning del LR: para Adam y Lion, hacer un sweep de LR [1e-5, 1e-2] log. Encontrar el LR óptimo de cada uno. Verificar que el de Lion es ~5× más chico.
3. AdamW vs Adam con L2: comparar Adam + keras.regularizers.L2(1e-2) en cada capa vs AdamW con weight_decay=1e-2. AdamW gana en val_loss.
4. Inspección de buffer: imprimir optimizer.variables. Adam tiene m y v por parámetro; Lion solo m. Verificar memoria total.
5. LR alto + Momentum: SGD con LR=0.1 explota; SGD+Momentum(0.9) con LR=0.1 puede funcionar. Probar.

Homework verificable

Sobre Fashion-MNIST + un MLP [300, 100, 10]:

1. Encontrar el LR óptimo para 3 optimizadores: SGD, AdamW, Lion (sweep de 5 valores cada uno).
2. Con el LR óptimo, entrenar 30 épocas y reportar val_accuracy.
3. Comparar wall time y memoria.

Criterio de aceptación: AdamW debe igualar o superar a Adam por ≥ 0.3 pp en val_accuracy; Lion con buen LR debe ser competitivo y consumir menos memoria del optimizer.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Cambiar de Adam a Lion sin cambiar LR	Lion necesita LR mucho más chico. Fix: div
Adam con weight_decay en Keras viejo	Lo aplica como L2 (mal). Fix: usar AdamW.
SGD plano sin schedule en redes profundas	Convergencia lentísima. Fix: SGD + Momentu
epsilon=1e-7 produce inestabilidad en bf16	Para mixed precision en LLMs, epsilon=1e-5
Asumir que Adam es siempre mejor	En visión clásica con suficiente data, SGD

Preguntas frecuentes

¿Adam o AdamW por default?

AdamW siempre que uses weight decay. Adam con weight_decay > 0 está mal implementado en muchos frameworks antiguos.

¿Lion en producción ya?

Sí, desde 2023. Google lo usa en internamente. Estable y bien probado.

¿Cuándo SGD gana a Adam?

Cuando podés permitirte LR + momentum + cosine schedule bien tuneados, sobre datasets grandes (ImageNet). El modelo final generaliza ~0.5-1 pp mejor. Pero requiere más tuning.

¿epsilon cuándo lo toco?

Casi nunca con fp32. En mixed precision (bf16/fp16), subir a 1e-4 para estabilidad.

¿beta_2 por qué 0.999 en Adam y 0.99 en Lion?

Adam el 2do momento debe ser estable (decay muy lento). Lion no tiene 2do momento — beta_2 define cómo se mezcla m_{t-1} con g_t en el lookahead, similar a Nesterov.

Referencias

- Géron, cap. 11 — Faster Optimizers.
- Kingma & Ba (2014), Adam, ICLR.
- Loshchilov & Hutter (2019), Decoupled Weight Decay Regularization (AdamW), ICLR.
- Chen et al. (2023), Symbolic Discovery of Optimization Algorithms (Lion), NeurIPS.
- Liu et al. (2023), Sophia: A Scalable Stochastic Second-order Optimizer.
- Keras optimizers.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 114 — Clase 114 — Optimizadores modernos: Lion, Sophia, Schedule-Free

> *Parte: 2 — Deep Learning · Fuente: Chen et al. (2023) Lion + Liu et al. (2023) Sophia + Defazio et al. (2024) Schedule-Free. Duración estimada: 75 min.*

Objetivo

Conocer la nueva generación de optimizadores 2023-2024 que está reemplazando a AdamW en LLM training a escala: Lion (Google, signo del gradiente), Sophia (Stanford, segundo orden aproximado), Schedule-Free (Meta, sin LR scheduling). Saber cuándo justifican el cambio.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar Lion con LR 3-10× más chico que AdamW y weight_decay 3-10× más grande.
- Aplicar Sophia con estimación diagonal del Hessiano (Hutchinson sampling).
- Aplicar Schedule-Free (schedulefree.AdamWScheduleFree) sin warmup/cosine.
- Comparar memoria, velocidad y calidad final.
- Reconocer cuándo Lion supera AdamW (modelos grandes, ViT, CLIP) y cuándo no.

Temas

- Lion: update = sign(β·m + (1-β)·g). 1 buffer en lugar de 2.
- Sophia: pre-condicionador diagonal del Hessiano vía Hutchinson.
- Schedule-Free: aprende sin schedule explícito, sin warmup.
- Memory: Lion ahorra 50 % vs AdamW.
- Trade-off: Lion + LR alto explota fácilmente.

Definiciones y características

- Lion: signo del gradiente como dirección. β=0.9, β=0.99 típicos.
- Sophia: actualiza θ ← θ - η·clip(m/h, -ρ, ρ). Robusto a hessianas mal condicionadas.
- Schedule-Free: averaging interpolation entre z (live) y x (averaged); no necesita LR schedule.
- Memory cost: Adam 2× params, AdamW 2× params, Lion 1× params, Schedule-Free 2× params.

Dataset / recursos

- Fashion-MNIST o CIFAR-10 + ViT-Tiny.
- Librerías: torch, torch.optim, schedulefree (pip), implementaciones Lion/Sophia community.

Ejercicios

1. AdamW baseline: ViT-Tiny en CIFAR-10. LR=1e-3, wd=0.05.
2. Lion: misma red, LR=1e-4, wd=0.5. Comparar accuracy y memoria.
3. Sophia: con Hutchinson cada 10 steps. Comparar convergencia.
4. Schedule-Free: AdamWScheduleFree(lr=1e-3, warmup_steps=500). Sin cosine.
5. Memory: para modelo grande, medir VRAM con cada uno.

Homework verificable

Comparar 4 optimizadores en ViT-Tiny + CIFAR-100:

1. AdamW (baseline).
2. Lion.
3. Sophia.
4. Schedule-Free AdamW.

Reportar: accuracy final, wall-time, peak VRAM.

Criterio de aceptación: Lion debe ahorrar ≥ 30 % VRAM vs AdamW; al menos uno de los modernos iguala o supera AdamW en accuracy.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Lion con LR de AdamW → loss=nan	LR debe ser 3-10× menor. Fix: 1e-4 a 3e-4
Sophia muy lento	Hutchinson costoso. Fix: estimar hessiano
Schedule-Free + cosine schedule	Conflicto. Fix: NO usar scheduler externo.
Comparar wall-time sin igual cantidad de e	Trampa. Fix: mismo epochs o early-stopping
Lion con weight_decay bajo	Resultados peores. Fix: subir wd 3-10× res

Preguntas frecuentes

Lion o AdamW en 2026?

Para modelos < 100M params: AdamW sigue siendo default seguro. Para LLM training a escala (>1B), Lion ahorra mucha memoria y gana papers (Chen 2023).

Sophia en producción?

Aún experimental. Google reportó 2× speedup en LLM pretraining (770M). Vale probar.

Schedule-Free realmente funciona sin warmup?

Recomienda warmup corto (500-1000 steps). El resto sin cosine.

Implementaciones?

Lion: implementaciones community (lucidrains/lion-pytorch). Sophia: similares. Schedule-Free: schedulefree package oficial Meta.

Lion en CV / fine-tuning?

Sí — paper original lo demostró en ViT, CLIP, modelos de difusión. Especialmente bueno para fine-tuning grande.

Referencias

- Chen et al. (2023), Symbolic Discovery of Optimization Algorithms (Lion), NeurIPS.
- Liu et al. (2023), Sophia: A Scalable Stochastic Second-order Optimizer.
- Defazio et al. (2024), The Road Less Scheduled (Schedule-Free).
- schedulefree.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 115 — Clase 115 — Learning rate scheduling

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 11 § Learning Rate Scheduling. Duración estimada: 60 min.*

Objetivo

Saber variar el LR durante el entrenamiento —no dejarlo fijo— porque ningún LR es óptimo en todas las fases. Aplicar las 4 estrategias estándar: step decay, exponential decay, cosine annealing (default moderno), y warmup + decay (estándar en Transformers).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Configurar keras.optimizers.schedules.CosineDecay y pasarlo como learning_rate= al optimizer.
- Diferenciar ExponentialDecay, PiecewiseConstantDecay y CosineDecayRestarts.
- Implementar warmup lineal + cosine — receta default en BERT/GPT.
- Usar ReduceLROnPlateau (reactivo) vs schedule (proactivo).
- Graficar la curva de LR a lo largo del entrenamiento para verificar.

Temas

- LR fijo: arrancás bien, terminás demasiado alto para refinar.
- Step decay: cortar LR cada N épocas. Simple, anticuado.
- Exponential decay: lr = lr_0 · γ^t.
- Cosine annealing (Loshchilov & Hutter 2017): lr = 0.5·lr_0·(1 + cos(πt/T)).
- Warmup: empezar bajo y subir linealmente las primeras X steps. Esencial en Transformers.
- One-cycle policy (Smith 2018): warmup + cosine descent + tail decay.

Definiciones y características

- Warmup: subir LR linealmente desde 0 (o muy bajo) hasta el lr_max en los primeros warmup_steps.

Estabiliza el inicio cuando los gradientes son ruidosos.

- Cosine annealing: bajar LR siguiendo una media curva coseno desde lr_max hasta lr_min. Suave, sin saltos.
- Restarts (SGDR): cada vez que llega al mínimo, reinicia con lr_max — exploración periódica.
- ReduceLROnPlateau: callback reactivo; baja LR cuando una métrica deja de mejorar.
- OneCycle: variante moderna que sube hasta lr_max en la primera mitad y baja en la segunda, terminando 100× por debajo del inicial.

Dataset / recursos

- Fashion-MNIST con un MLP medio.
- Librerías: tensorflow, keras, matplotlib.

Ejercicios

1. Schedule básica: lr = CosineDecay(initial_learning_rate=1e-3, decay_steps=10_000); Adam(learning_rate=lr). Entrenar y graficar val_loss.
2. Visualizar el LR: para una schedule, evaluarla en steps 0, 100, 1000, 5000, 10000 y graficar.
3. Warmup + Cosine: implementar custom callback (o usar CosineDecay(warmup_steps=...) en Keras 3) y entrenar un Transformer chico (anticipo). Comparar contra sin warmup.
4. ReduceLROnPlateau: alternativa reactiva — ReduceLROnPlateau(factor=0.5, patience=3). Comparar con cosine.
5. One-cycle: implementar con LearningRateScheduler callback. Probar y comparar.

Homework verificable

Sobre Fashion-MNIST:

1. Entrenar 3 modelos con la misma arquitectura: (a) LR fijo 1e-3, (b) ExponentialDecay, (c) CosineDecay(warmup_steps=100).
2. Reportar val_accuracy y graficar las curvas de loss + LR.
3. Concluir qué schedule produjo mejor accuracy.

Criterio de aceptación: cosine con warmup debe ganar o empatar contra fijo. La curva de val_loss del schedule debe ser visualmente más suave hacia el final.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Schedule decae demasiado rápido	decay_steps está mal calibrado vs total st
Combinar schedule + ReduceLROnPlateau	Conflicto: ambos modifican LR. Fix: elegí
LearningRateScheduler callback no funciona	Conflicto. Fix: usar uno u otro.
Sin warmup, transformer diverge en las pri	Gradientes iniciales son ruidosos. Fix: wa
Reiniciar entrenamiento desde checkpoint p	optimizer.iterations cuenta steps; al reca

Preguntas frecuentes

¿Cuál schedule por default?

Cosine con warmup. Es el estándar 2022+ en visión y NLP.

¿Cuántos warmup steps?

5-10 % del total. Para 100 épocas de 500 steps cada una = 50 000 steps → warmup 2 500-5 000.

¿Schedule en steps o en épocas?

Steps casi siempre. Cuanto más granular, más suave.

¿CosineDecayRestarts cuándo?

Cuando entrenás muy largo y querés exploraciones periódicas. Útil en redes muy profundas y datasets grandes; raro en proyectos chicos.

¿Schedule depende del optimizer?

Casi no — Adam(lr=schedule) funciona igual que SGD(lr=schedule). La diferencia es en los valores absolutos.

Referencias

- Géron, cap. 11 — Learning Rate Scheduling.
- Loshchilov & Hutter (2017), SGDR: Stochastic Gradient Descent with Warm Restarts, ICLR.
- Smith (2018), Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates.
- Keras LR schedules.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 116 — Clase 116 — Regularización: L1/L2, dropout, max-norm, MC dropout (+ Stochastic Depth, DropPath)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 11 § Regularization + Huang et al. (2016) Deep Networks with Stochastic Depth. Duración estimada: 80 min.*

Objetivo

Conocer las técnicas de regularización en DL —L1/L2, dropout (Srivastava et al. 2014), max-norm, MC dropout para incertidumbre— y las técnicas modernas que se usan en arquitecturas profundas (ResNets, ViT, Transformers): Stochastic Depth, DropPath y LayerDrop.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar keras.regularizers.l1(...), l2(...), l1_l2(...) en una capa.
- Aplicar Dropout(rate=0.5) y entender qué hace en train vs en inference (default desactivado).
- Implementar Monte Carlo dropout (Dropout(0.5) activo en inference → predicciones diferentes → incertidumbre).
- Aplicar Stochastic Depth en una ResNet: dropear bloques residuales completos al azar durante training.
- Aplicar DropPath (estándar en ViT, Swin Transformer, ConvNeXt).

Temas

- L1/L2 como penalización en la loss. λ típicamente 1e-4 a 1e-2.
- Dropout: enmascarar fracción r de las activaciones por batch.

- Inverted dropout: en inference no se hace nada porque train ya escala por 1/(1-r).
- Max-norm constraint: ||w|| ≤ c por neurona después de cada update.
- MC Dropout (Gal & Ghahramani 2016): incertidumbre bayesiana aproximada.
- Complemento moderno: Stochastic Depth, DropPath (= Stochastic Depth aplicado a paths de attention/FFN), LayerDrop (Fan et al. 2020).

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 104b — Regularización moderna: Stochastic Depth, DropPath, LayerDrop

Definiciones y características

- L1 regularization: agrega λ·Σ|w| a la loss. Promueve sparsity.
- L2 regularization (weight decay): agrega λ·Σw². Mantiene pesos chicos.
- Dropout: enmascara fracción r de neuronas por batch. Forzar redundancia.
- MC Dropout: hacer N predicciones con dropout activo → distribución de predicciones → incertidumbre.
- Max-norm: constraint sobre la norma de los pesos por unidad.
- Stochastic Depth: dropear bloques residuales enteros durante training.
- DropPath: como Stochastic Depth pero para paths en transformer (attention o FFN).

Dataset / recursos

- Fashion-MNIST + un MLP propenso a overfit.
- Librerías: tensorflow, keras, matplotlib.

Ejercicios

1. Sin regularización: entrenar un MLP grande ([512, 256, 128]) en Fashion-MNIST y observar overfitting (gap train/val ≥ 5 pp).
2. L2: agregar kernel_regularizer=keras.regularizers.l2(1e-3) a cada Dense. Comparar.
3. Dropout: agregar Dropout(0.3) entre Dense layers. Comparar.
4. MC Dropout: para 1 sample de test, hacer 100 predicciones con model(x, training=True). Calcular mean ± std de las probabilidades. Interpretar la incertidumbre.
5. Stochastic Depth simulado: en un mini ResNet con 8 bloques, dropear cada bloque con prob 0.1 lineal. Comparar contra sin stochastic depth.

Homework verificable

Sobre Fashion-MNIST con MLP [512, 256, 128, 64]:

1. Entrenar 4 versiones: sin regularización; L2(1e-3); Dropout(0.3); L2 + Dropout combinados.
2. Reportar train_acc y val_acc; calcular el gap.
3. Para el mejor modelo, hacer MC Dropout con 50 muestras sobre 5 imágenes ambiguas y reportar incertidumbre.

Criterio de aceptación: el modelo regularizado tiene gap train-val menor a 3 pp (vs ~6 pp del baseline) y val_acc igual o mejor. MC dropout debe asignar mayor std a las imágenes ambiguas.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Dropout en inferencia da resultados distin	Pasaste training=True por error. Fix: en i

L2 con λ=1.0 y modelo no aprende	Penalización demasiado fuerte. Fix: λ típi
Dropout(0.5) en la última capa antes de so	Distorsiona logits. Fix: dropout en capas
L2 + AdamW con weight_decay → doble penali	Usar uno: AdamW(wd=...) o kernel_regulariz
Stochastic Depth con p_i constante en luga	Funciona pero menos óptimo. Fix: p_i = i/N

Preguntas frecuentes

¿Dropout 0.5 siempre?

0.5 para capas Dense grandes. Para capas Conv: 0.1-0.2. Para embeddings y attention en Transformers: 0.1.

¿BN ya regulariza, necesito dropout también?

Depende. En CNNs/MLPs con BN, dropout a veces ya no aporta. En Transformers, sí (BN no se usa allí; LN + dropout + DropPath).

¿MC Dropout es bayesiano "de verdad"?

Aproxima un proceso gaussiano variacional. No es bayesiano riguroso pero es una excelente aproximación práctica para incertidumbre.

¿Stochastic Depth en CNN no residual?

No tiene sentido — Stochastic Depth necesita la skip connection para que dropear no rompa el forward.

¿Cuánta dropout/droppath en ViT base?

ViT-Base original: dropout=0.1 en attention, droppath=0.1 lineal en cada bloque. Para fine-tuning, suele bajarse a 0.0.

Referencias

- Géron, cap. 11 — Regularization Using Dropout.
- Srivastava et al. (2014), Dropout, JMLR.
- Gal & Ghahramani (2016), Dropout as a Bayesian Approximation, ICML — MC dropout.
- Huang et al. (2016), Deep Networks with Stochastic Depth, ECCV.
- Fan et al. (2020), Reducing Transformer Depth on Demand with Structured Dropout (LayerDrop).
- keras DropPath / StochasticDepth.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 117 — Clase 117 — Regularización moderna: Stochastic Depth, DropPath, LayerDrop

> *Parte: 2 — Deep Learning · Fuente: Huang et al. (2016) Stochastic Depth + Fan et al. (2020) LayerDrop + DropPath en ViT/Swin/ConvNeXt. Duración estimada: 70 min.*

Objetivo

Aplicar regularización por paths/bloques —más allá del dropout clásico— en arquitecturas profundas

modernas (ResNet, ViT, ConvNeXt, Swin Transformer). Cubrir Stochastic Depth (drop bloque residual), DropPath (drop path en transformer), LayerDrop (drop layer completa). Beneficio doble: regularización + reducción de cómputo durante training.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar keras.layers.StochasticDepth(rate) o DropPath(rate) en bloques residuales.
- Diseñar rate lineal por profundidad: capa 0 → 0.0, capa N → 0.2.
- Aplicar LayerDrop durante pretraining para permitir inference con menos capas.
- Diferenciar Dropout (neurona) vs Stochastic Depth (bloque) vs LayerDrop (layer).
- Reconocer el speedup de training: 25 % más rápido en ResNet-110 (Huang 2016).

Temas

- Dropout clásico vs Stochastic Depth.
- DropPath = Stochastic Depth para Transformers.
- Rate lineal: p_i = i/N · p_max.
- LayerDrop para compresión de Transformers.
- Combinaciones: Dropout + DropPath + Label Smoothing.

Definiciones y características

- Stochastic Depth: y = x + drop(b(x)) con prob p_i. En inference, scale por 1 - p_i.
- DropPath: igual idea aplicada a attention y FFN paths.
- LayerDrop: dropear layer entera del Transformer.
- Linear rate schedule: capas tempranas seguras (p≈0), tardías regularizadas (p≈0.2).

Dataset / recursos

- CIFAR-10/100 con ResNet propia.
- ViT-Tiny preentrenado.
- Librerías: torch, timm (donde DropPath es default).

Ejercicios

1. ResNet con Stochastic Depth: implementar BasicBlock con StochasticDepth(p). Train CIFAR-10.
2. Rate lineal: aplicar p_i = i/N · 0.2 en cada bloque. Comparar contra rate constante.
3. ViT con DropPath: timm.create_model('vit_tiny_patch16_224', drop_path_rate=0.1). Comparar contra 0.0.
4. LayerDrop: simular con 12-layer BERT mini — drop 50 % layers. Verificar accuracy aún razonable.
5. Speed: medir wall-clock training con vs sin Stochastic Depth.

Homework verificable

ResNet-50 en CIFAR-100:

1. Entrenar con y sin Stochastic Depth (rate lineal a 0.2 final).
2. Reportar accuracy + tiempo total.
3. Verificar regularización: gap train-val.

Criterio de aceptación: Stochastic Depth reduce gap train-val ≥ 1 pp; speedup en wall-time visible (~10-20 %).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar

Rate constante 0.5 en todas las capas	Mata bloques iniciales (críticos). Fix: li
Olvido scale en inference	Sesgo. Fix: librerías lo manejan; verifica
DropPath sin residual	Sin sentido — no hay path para droppear. F
Aplicar a 1ª y última capa	Generalmente innecesario. Fix: layers inte
Combinar Dropout + Stochastic Depth + Labe	Overregularization. Fix: ajustar individua

Preguntas frecuentes

Stochastic Depth o Dropout?

Para CNN/ViT profundas: ambos. Stochastic Depth en bloques residuales, Dropout en MLP final.

Rate final 0.1 o 0.2 o 0.5?

0.1-0.2 para ResNet/ViT base. 0.3-0.5 para modelos enormes (LayerDrop en BERT-Large).

LayerDrop sirve para inference?

Sí — train con LayerDrop=0.5, inference con k random layers. Habilita modelos compactos sin re-training.

timm vs implementación propia?

Usá timm siempre que se pueda — DropPath bien implementado, rate scheduling automático.

En LLMs modernos (Llama)?

Less común. Llama no usa DropPath. BERT/RoBERTa sí.

Referencias

- Huang et al. (2016), Deep Networks with Stochastic Depth, ECCV.
- Fan et al. (2020), Reducing Transformer Depth on Demand with Structured Dropout (LayerDrop), ICLR.
- Dosovitskiy et al. (2020), ViT — DropPath aplicado.
- timm DropPath.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 118 — Clase 118 — TensorFlow: tensores, variables, operaciones

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 12 § Using TensorFlow like NumPy. Duración estimada: 60 min.*

Objetivo

Bajar un nivel por debajo de Keras: trabajar directamente con tensores (tf.Tensor) y variables (tf.Variable), entender la API NumPy-like de TF (tf.matmul, tf.reduce_*, tf.cast, tf.reshape), y diferenciar inmutable (Tensor) de mutable (Variable, base de los pesos).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Crear tensores con tf.constant, tf.zeros, tf.ones, tf.random.normal.
- Aplicar operaciones: aritméticas, matmul, broadcasting, indexing/slicing, reduce_mean/sum/max.
- Convertir entre TF y NumPy (.numpy(), tf.convert_to_tensor).
- Crear y modificar tf.Variable con .assign, .assign_add.
- Reconocer dtypes (float32, float64, int32, bool) y forzar cast cuando hace falta.

Temas

- tf.Tensor: inmutable, similar a np.ndarray.
- tf.Variable: mutable, base de los pesos.
- Broadcasting (idéntico a NumPy).
- Operaciones reduce con axis=.
- tf.function (anticipo clase 107) — convierte una función Python en grafo.
- TF en GPU: ops sobre tensores van a GPU automáticamente si está disponible.

Definiciones y características

- tf.constant: crea un Tensor inmutable.
- tf.Variable: tensor mutable, registrable como peso de un modelo.
- Broadcasting: (3, 1) + (1, 4) → (3, 4). Reglas idénticas a NumPy.
- dtype: tipo numérico. Mismatchs disparan errores; usar tf.cast(x, tf.float32).
- .assign(): actualizar el contenido de una Variable in-place. v.assign_add(g) es v += g.
- .shape y .numpy(): forma del tensor y conversión a array NumPy.

Dataset / recursos

- Operaciones a mano (no requiere dataset).
- Librerías: tensorflow, numpy.

Ejercicios

1. Tensores básicos: crear t = tf.constant([[1.0, 2.0], [3.0, 4.0]]). Imprimir shape, dtype, t.numpy().
2. Operaciones: tf.matmul(t, t), tf.transpose(t), tf.reduce_sum(t, axis=0). Verificar shapes.
3. Broadcasting: a = tf.constant([[1.], [2.], [3.]]) (shape 3,1), b = tf.constant([10., 20., 30.]) (3,). a + b → ¿qué shape?
4. Variable y assign: v = tf.Variable([1., 2., 3.]); v.assign([4., 5., 6.]); v.assign_add([1., 1., 1.]). Verificar.
5. dtype mismatch: tf.constant([1, 2, 3]) + tf.constant([1.0, 2.0, 3.0]) → error. Arreglar con tf.cast.

Homework verificable

Implementar regresión lineal manualmente con TF (sin Keras):

1. Generar X = tf.random.normal((100, 2)), y_true = X @ tf.constant([[1.], [2.]]) + 3 + ruido.
2. Variables w = tf.Variable(tf.random.normal((2, 1))), b = tf.Variable(0.0).
3. Loss = MSE.
4. Loop manual: calcular pred, loss, gradientes (con GradientTape — anticipo clase 108), update con assign_sub.
5. Reportar w, b finales, comparar con verdaderos.

Criterio de aceptación: tras 500 iteraciones con LR=0.01, w ≈ [1, 2] (±0.1) y b ≈ 3 (±0.1).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
InvalidArgumentError: cannot compute Add a	dtype mismatch. Fix: tf.cast(x, tf.float32

AttributeError: 'EagerTensor' object has n	Dentro de @tf.function los tensores son si
Asignar a tf.constant	Inmutable. Fix: usar tf.Variable.
Operaciones tipo arr[mask] = 0 en TF	TF tensors son inmutables. Fix: tf.where(m
t.numpy() lento dentro de un loop de entre	Forza sync GPU → CPU. Fix: mantener todo e

Preguntas frecuentes

¿TF tensors o NumPy arrays?

TF cuando trabajás con un modelo y querés que las operaciones corran en GPU + sean diferenciables. NumPy para análisis CPU de datos. Conversión es barata pero no gratis.

¿tf.Variable o tf.constant para datos de entrada?

Constant (los datos no cambian). Variables son para pesos que se entrenan.

¿Por qué float32 y no float64?

GPUs son MUCHO más rápidas en float32 (y aún más en bfloat16). Default ML.

¿Cómo seteo el device manualmente?

with tf.device('/GPU:0'): ... o tf.config.set_visible_devices(...). En la mayoría de los casos no hace falta — TF lo elige bien.

¿Equivalencia con PyTorch?

tf.constant ↔ torch.tensor(..., requires_grad=False). tf.Variable ↔ torch.Parameter o nn.Parameter. Operaciones casi 1:1.

Referencias

- Géron, cap. 12 — Custom Models and Training with TensorFlow.
- TF Tensor guide.
- TF Variable guide.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 119 — Clase 119 — Losses, métricas, capas, modelos custom

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 12 § Customizing Models and Training Algorithms. Duración estimada: 70 min.*

Objetivo

Crear losses, métricas y capas custom cuando los builtins de Keras no alcanzan: focal loss, métrica F1 macro, una capa con normalización custom, modelo subclassed con train_step propio. Diferenciar stateless (función) de stateful (clase con estado acumulado por época).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Definir loss custom como función: def my_loss(y_true, y_pred): return ....
- Definir métrica stateful heredando keras.metrics.Metric con update_state, result, reset_state.
- Crear capa custom heredando keras.layers.Layer con build (declara pesos) y call (forward).
- Overridar train_step de un modelo subclassed (Model) para custom training logic.
- Saber cuándo usar custom vs cuándo los builtins de Keras alcanzan (casi siempre).

Temas

- Loss como función simple: 2 args (y_true, y_pred), devuelve un tensor.
- Métrica stateless (función) vs stateful (Metric class).
- Capa custom: __init__ (config), build (pesos), call (forward).
- Model subclass con train_step(self, data) custom.

Definiciones y características

- Loss: función que devuelve un escalar (o vector por sample) que minimizamos.
- Métrica stateful: acumula a lo largo del epoch (necesario para F1, precision, recall, AUC).
- build(input_shape): hook donde Keras te dice las dims; ahí declarás pesos con self.add_weight.
- get_config(): serialización — necesario si querés model.save().
- train_step: lo que Keras llama por cada batch durante fit. Default es: forward, compute loss, backward, optimizer step.

Dataset / recursos

- Fashion-MNIST o cualquier dataset previo.
- Librerías: tensorflow, keras.

Ejercicios

1. Loss custom: implementar Focal Loss FL(p_t) = -α(1-p_t)^γ log(p_t) (Lin et al. 2017, útil para imbalance). Aplicar a Fashion-MNIST y comparar contra cross-entropy.
2. Métrica F1 macro: heredar keras.metrics.Metric, mantener confusion matrix acumulada por época, calcular F1 macro en result().
3. Capa custom: class L2Normalize(Layer) que normaliza cada vector a norma 1. Probarla en un modelo.
4. Modelo con train_step custom: subclass que en cada batch aplica gradient clipping manual + logging extra.
5. get_config: agregar a una capa custom; verificar que model.save() y load_model(custom_objects=...) funciona.

Homework verificable

Sobre un dataset desbalanceado (simular o usar uno con clase minoritaria al 5 %):

1. Entrenar con cross-entropy estándar; reportar F1 macro.
2. Entrenar con Focal Loss custom (γ=2.0, α=0.25); reportar F1.
3. Implementar una métrica F1 macro custom y usarla durante el training.

Criterio de aceptación: Focal Loss debe mejorar F1 macro en ≥ 2 pp respecto a cross-entropy en el caso desbalanceado.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar

Loss custom devuelve shape incorrecto	Debe ser (batch,) o escalar. Fix: revisar
Métrica custom devuelve siempre el mismo v	Olvidaste reset_state entre épocas. Keras
model.save() falla con capa custom	Falta get_config. Fix: implementarlo y cls
Pesos creados en __init__ en vez de build	Funciona pero pierde la flexibilidad de sh
Override train_step y se pierde el logging	Hay que actualizar self.compiled_metrics.u

Preguntas frecuentes

¿Custom loss o tf.keras.losses.Loss subclass?

Función para casos simples; subclass cuando tenés estado o configs complejas.

¿Métrica función vs subclass?

Función para batch-wise (accuracy). Subclass para cualquier cosa que necesite acumular (precision, recall, F1, AUC).

¿call o __call__ en capa custom?

Definí call; Keras envuelve para llamar a través de __call__ agregando training/masking.

¿Custom training loop o subclass con train_step?

Subclass train_step cuando podés (preserva fit, callbacks, etc.). Custom loop completo solo si necesitás control de flujo realmente complejo (clase 108).

¿Por qué add_weight y no tf.Variable?

add_weight registra automáticamente la variable como peso entrenable del modelo, gestiona dtype, initializer, regularizer y nombre.

Referencias

- Géron, cap. 12 — Custom Loss Functions, Metrics, Layers and Models.
- Lin et al. (2017), Focal Loss for Dense Object Detection, ICCV.
- Keras — Making new layers and models.
- Keras — Customize what happens in fit().

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 120 — Clase 120 — Funciones y grafos (autograph)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 12 § TF Functions and Graphs. Duración estimada: 55 min.*

Objetivo

Entender qué hace @tf.function —compila una función Python a un grafo TF estático, acelerando 2-10× y permitiendo deploy en TF Serving / TFLite—. Conocer AutoGraph (traduce automáticamente if/for/while Python a operaciones TF), saber los gotchas clásicos (efectos colaterales, prints, listas Python) y cuándo el

decorator deteriora la experiencia de debugging.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar @tf.function a una función custom y verificar speedup.
- Identificar cuándo NO usarlo (debugging, lógica con efectos colaterales no determinísticos).
- Entender retracing: cada vez que cambias la shape o dtype del input, TF reconstruye el grafo.
- Usar tf.function(input_signature=...) para evitar retracing.
- Diferenciar eager mode (default, dinámico) de graph mode (compilado).

Temas

- Eager vs graph execution.
- @tf.function y AutoGraph.
- Retracing: por qué pasar Python ints vs tensors causa retraces.
- Print en grafo: tf.print (corre en graph) vs print (solo en tracing).
- Cuándo @tf.function vale la pena (loops, training step) y cuándo no (one-shot, debugging).

Definiciones y características

- Eager mode: cada op se ejecuta inmediatamente. Default en TF 2+. Como NumPy.
- Graph mode: las ops se ensamblan en un grafo y se ejecutan después. Más rápido, deployable.
- @tf.function: convierte una función Python en un ConcreteFunction (grafo).
- AutoGraph: subsistema que traduce if/for/while a tf.cond/tf.while_loop.
- Retracing: rebuild del grafo cuando cambian shape/dtype de inputs. Caro.
- input_signature: tupla de TensorSpec que fija las shapes esperadas → no retracing.

Dataset / recursos

- Operaciones aisladas para medir velocidad.
- Librerías: tensorflow, time.

Ejercicios

1. Speedup básico: definir def f(x): return tf.reduce_sum(x * 2 + 3x + 1). Medir tiempo eager vs @tf.function-wrapped en un loop de 10 000 iteraciones.
2. Retracing: definir una función con @tf.function. Llamarla con tensores de shapes distintas; usar tf.config.experimental_run_functions_eagerly(True) y print para detectar retraces.
3. AutoGraph: una función con un for Python y un if. Verificar que se convierte correctamente (tf.autograph.to_code(f)).
4. tf.print vs print: dentro de @tf.function, demostrar que print solo se ejecuta en tracing (1ª llamada), tf.print siempre.
5. input_signature: fijar input_signature para evitar retracing con shape (None, 784).

Homework verificable

Implementar un training loop minimalista (sin Keras) con y sin @tf.function:

1. Modelo simple en numpy/TF (regresión lineal).
2. Loop manual de 1 000 iteraciones.
3. Mismo loop wrapped con @tf.function.
4. Comparar tiempo total.

Criterio de aceptación: la versión @tf.function debe ser ≥ 2× más rápida (en CPU, más en GPU).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
print no aparece dentro de @tf.function de	Solo corre en tracing. Fix: tf.print para
Función retrace en cada llamada	Pasás Python ints/strings como argumentos.
list.append() dentro de @tf.function no fu	Listas Python no se preservan en grafo. Fi
if x > 0: con tensor x → error o warning	AutoGraph debería convertirlo a tf.cond. F
np.random dentro de @tf.function siempre d	Se evalúa una vez en tracing. Fix: usar tf

Preguntas frecuentes

¿Cuándo @tf.function?

Para funciones que se llaman muchas veces (training step, custom layers complejas). NO para funciones one-shot, debugging, o muy simples.

¿@tf.function y debugging?

Hacé el debugging en eager (sin decorator). Una vez que funciona, agregás @tf.function para producción.

¿Cómo veo el grafo generado?

tf.autograph.to_code(f.python_function) te da el Python equivalente que generó AutoGraph. Útil para entender qué hizo.

¿jit_compile=True qué hace?

@tf.function(jit_compile=True) activa XLA, otra capa de optimización (fusión de ops). Mejora velocidad otro 1.5-3×, especialmente en TPU.

¿En Keras hace falta?

No para model.fit (ya está jitteado). Sí para custom training loops (clase 108) y para custom layers/models complejos.

Referencias

- Géron, cap. 12 — TF Functions and Graphs.
- TF — Better performance with tf.function.
- TF — AutoGraph.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 121 — Clase 121 — Custom training loops (+ PyTorch & PyTorch Lightning)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 12 § Custom Training Loops + docs PyTorch, Lightning. Duración estimada: 85 min.*

Objetivo

Escribir un training loop manual en TF con GradientTape — control absoluto sobre cada paso (útil para GANs, RL, multi-step optimizers, debugging). Conocer el equivalente en PyTorch (el framework dominante de la industria en 2026) y cómo PyTorch Lightning abstrae los boilerplate del loop, devolviendo la productividad de Keras con la flexibilidad de PyTorch.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Escribir un training loop TF con for batch in dataset: with GradientTape() as tape: ...; grads = tape.gradient(loss, vars); optimizer.apply_gradients(...).
- Hacer el equivalente en PyTorch: optimizer.zero_grad(); loss.backward(); optimizer.step().
- Usar PyTorch Lightning para el mismo problema con boilerplate mínimo (LightningModule.training_step).
- Reconocer cuándo el loop manual es necesario (modelo con dos optimizadores, schedule custom step-wise, RL).
- Comparar productividad y flexibilidad de los 3 frameworks.

Temas

- Loop básico TF: epochs → batches → tape → grads → apply.
- tape.watch(...) para watch tensors que no son tf.Variable.
- Métricas y logging manual.
- Equivalente PyTorch.
- Lightning como capa de abstracción.
- Complemento moderno: PyTorch + Lightning en paralelo a TF/Keras.

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 108a — PyTorch fundamentos: tensores, autograd, nn.Module
- Clase 108b — PyTorch Lightning: Trainer + distributed

Definiciones y características

- GradientTape: contexto TF que registra operaciones para autodiff.
- tape.watch(t): forzar a tape a registrar un tensor no-Variable.
- tape.gradient(loss, vars): calcular gradientes.
- optimizer.apply_gradients(zip(grads, vars)): aplicar la update.
- PyTorch .backward(): aplica autodiff sobre el grafo dinámico construido en el forward.
- requires_grad: flag PyTorch que indica si un tensor necesita gradientes.

Dataset / recursos

- Fashion-MNIST o MNIST.
- Librerías: tensorflow, torch, lightning (pip install lightning).

Ejercicios

1. TF loop manual: entrenar un MLP en Fashion-MNIST con GradientTape. Implementar logging de loss y métrica manual.
2. PyTorch equivalente: reimplementar el mismo loop en PyTorch.
3. Lightning: mismo problema con LightningModule.
4. Speedup con jit: tf.function en TF; torch.compile en PyTorch. Medir.

5. Multi-optimizer: con TF, escribir un loop que aplica un optimizer para las capas frozen-ish (LR bajo) y otro para las nuevas (LR alto). Esto en model.fit requiere mucho boilerplate.

Homework verificable

Implementar el mismo problema (Fashion-MNIST, MLP [256, 128]) en los 3 frameworks:

1. TF con custom training loop.
2. PyTorch puro.
3. PyTorch Lightning.

Reportar para cada uno: # líneas de código, tiempo de training, accuracy.

Criterio de aceptación: las 3 versiones llegan a accuracy similar (±0.5 pp). Lightning ≈ Keras en LOC; PyTorch puro tiene más boilerplate; TF custom loop también.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Loss no baja en TF custom loop	Olvidaste tape.gradient(loss, model.traina
En PyTorch, gradientes acumulados batch a	Olvidaste optimizer.zero_grad() al inicio
RuntimeError: cudnn error en PyTorch	Versión de PyTorch incompatible con CUDA i
Lightning espera LightningDataModule cuand	Lightning acepta ambos; verificar firma de
BatchNorm no se actualiza en TF custom loo	Hay que pasar training=True explícitamente

Preguntas frecuentes

¿TF o PyTorch para empezar?

Keras (TF) es más amigable para empezar. PyTorch es más demandado en producción 2026. Saber ambos es el estándar profesional.

¿Lightning vs Keras de cuál sale mejor?

Lightning te abre el ecosistema PyTorch (Hugging Face). Keras 3 ahora soporta backends multi (TF, JAX, PyTorch) — la diferencia disminuye.

¿zero_grad() por qué existe en PyTorch?

Para permitir gradient accumulation (varios .backward() sin step para batches grandes virtuales). Si no necesitás eso, llamá zero_grad() al inicio del batch siempre.

¿Distributed training más fácil en cuál?

Lightning, por mucho. trainer = L.Trainer(strategy='ddp', devices=4) y ya. En TF necesitás tf.distribute.MirroredStrategy con strategy.scope():. PyTorch puro requiere torch.distributed.init_process_group manual.

¿Hugging Face usa PyTorch o TF?

Casi todo PyTorch desde 2022. Los modelos modernos (Llama, Mistral, Mixtral, etc.) están solo en PyTorch.

Referencias

- Géron, cap. 12 — Custom Training Loops.
- PyTorch tutorials.
- PyTorch Lightning docs.

- Falcon et al. (2019), PyTorch Lightning.
- Hugging Face — Trainer (built on Lightning concepts).

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 122 — Clase 122 — PyTorch fundamentos: tensores, autograd, nn.Module

> *Parte: 2 — Deep Learning · Fuente: PyTorch tutorials + Howard & Gugger, Deep Learning for Coders with fastai & PyTorch. Duración estimada: 90 min.*

Objetivo

Aprender PyTorch —el framework dominante en research y en LLMs/multimodal 2026—. Cubrir: tensores (similar a NumPy, en GPU), autograd (requires_grad, .backward()), nn.Module (forma de definir modelos), Dataset/DataLoader para data pipelines. Equivalencias 1:1 con Keras/TF de las clases anteriores.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Crear tensores: torch.tensor, torch.zeros, torch.randn, device='cuda'.
- Aplicar autograd: x.requires_grad_(True); y = f(x); y.backward(); x.grad.
- Definir un MLP custom: class Net(nn.Module): def __init__(self): ...; def forward(self, x): ....
- Escribir el loop manual: optimizer.zero_grad(); loss.backward(); optimizer.step().
- Usar Dataset y DataLoader para pipelines de datos.

Temas

- Tensors vs ndarray, .to(device).
- Computation graph dinámico (vs estático TF1) — define-by-run.
- requires_grad y autograd.
- nn.Module, nn.Linear, nn.Sequential, nn.functional.
- Loss funcs: nn.CrossEntropyLoss, nn.MSELoss.
- Optim: torch.optim.Adam(model.parameters(), lr=...).
- Dataset + DataLoader(num_workers, pin_memory).

Definiciones y características

- torch.tensor: similar a np.ndarray pero con GPU + autograd.
- nn.Module: clase base de todos los modelos. __init__ declara capas, forward define forward.
- nn.Parameter: tensor con requires_grad=True registrado automáticamente.
- optimizer.zero_grad(): OBLIGATORIO al inicio de cada batch (sino se acumulan).
- model.train() / model.eval(): cambia comportamiento de BatchNorm y Dropout.
- with torch.no_grad():: contexto para inference, no construye grafo.

Dataset / recursos

- Fashion-MNIST vía torchvision.datasets.
- Librerías: torch, torchvision.

Ejercicios

1. Tensores: crear, mover a GPU, operaciones básicas. Comparar con NumPy.
2. Autograd: x = torch.tensor([2.0], requires_grad=True); y = x**3; y.backward(); print(x.grad) → debe ser 12.
3. MLP custom: definir clase con 2 nn.Linear + ReLU. Verificar model.parameters().
4. Training loop manual: Fashion-MNIST, 1 época, reportar loss.
5. DataLoader: DataLoader(dataset, batch_size=32, shuffle=True, num_workers=2). Iterar.

Homework verificable

Reproducir el modelo de Fashion-MNIST de Clase 091 en PyTorch:

1. MLP [300, 100, 10].
2. CrossEntropy loss, Adam(1e-3).
3. EarlyStopping manual (cuando val_loss no baja por 5 épocas).
4. Reportar test accuracy.

Criterio de aceptación: accuracy ≥ 0.87 (igual que la versión Keras); código compacto y legible.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Olvido optimizer.zero_grad()	Gradientes acumulados. Fix: agregar al ini
Modelo no aprende	Olvidaste .to(device) en model o data. Fix
RuntimeError: Trying to backward through t	Reusar el grafo. Fix: loss.backward(retain
Eval con model.train() (BN/Dropout activos	Métricas distorsionadas. Fix: model.eval()
DataLoader lento	num_workers=0. Fix: num_workers=4, pin_mem

Preguntas frecuentes

¿PyTorch o TF/Keras?

PyTorch para research, LLMs, multimodal, HuggingFace ecosystem. Keras para tabular/imagen estándar. Saber ambos.

¿torch.compile mejora velocidad?

Sí — model = torch.compile(model) en PyTorch 2.0+: 2-5× speedup automático sin cambiar código.

¿Equivalente de Keras fit?

PyTorch puro no lo tiene. Lo provee Lightning (clase 108b) y HuggingFace Trainer.

¿nn.Sequential o nn.Module custom?

Sequential para pilas lineales. Custom Module para cualquier topología no trivial.

¿GPU mac (Apple Silicon)?

device='mps' desde PyTorch 1.12. Funcional, ~50-80 % de NVIDIA performance.

Referencias

- PyTorch tutorials.
- Howard & Gugger, Deep Learning for Coders with fastai & PyTorch.
- Paszke et al. (2019), PyTorch: An Imperative Style, High-Performance Deep Learning Library, NeurIPS.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 123 — Clase 123 — PyTorch Lightning: Trainer, callbacks, distributed

> *Parte: 2 — Deep Learning · Fuente: Lightning docs. Duración estimada: 80 min.*

Objetivo

Aprender PyTorch Lightning — la capa de abstracción que convierte PyTorch puro (mucho boilerplate) en algo tan productivo como Keras pero conservando flexibilidad. Cubrir LightningModule, Trainer, callbacks, logging (W&B/TensorBoard), distributed training con un solo kwarg, mixed precision automática.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Subclassear LightningModule con training_step, validation_step, configure_optimizers.
- Usar Trainer(max_epochs, accelerator='auto', devices='auto', precision='bf16-mixed', logger=...).
- Aplicar callbacks: EarlyStopping, ModelCheckpoint, LearningRateMonitor.
- Activar distributed con strategy='ddp' o 'fsdp' para multi-GPU sin reescribir nada.
- Loggear a W&B / TensorBoard / MLflow vía 1 línea.

Temas

- LightningModule vs nn.Module puro.
- Trainer args: max_epochs, devices, precision, strategy, accumulate_grad_batches.
- Callbacks integrados.
- LightningDataModule para data pipelines.
- Distributed training: ddp, fsdp, deepspeed.
- Logging multi-backend.

Definiciones y características

- LightningModule: extiende nn.Module con hooks (training_step, etc.).
- Trainer: orquesta el entrenamiento. Maneja loops, device, distributed.
- self.log(...): registra métrica al logger; se agrega en epoch/batch.
- strategy: 'auto' | 'ddp' | 'fsdp' | 'deepspeed_stage_2'.
- precision: '32-true' | '16-mixed' | 'bf16-mixed'.

Dataset / recursos

- Fashion-MNIST o cualquier dataset previo.
- Librerías: lightning, torch, torchmetrics.

Ejercicios

1. LightningModule básico: convertir el MLP de 108a a Lightning.
2. Callbacks: agregar EarlyStopping(patience=5) + ModelCheckpoint(save_top_k=3).
3. Mixed precision: Trainer(precision='bf16-mixed'). Comparar tiempo.
4. W&B logging: Trainer(logger=WandbLogger(project='test')). Ver curvas online.

5. DDP: si tenés 2+ GPUs, strategy='ddp', devices=2. Verificar speedup.

Homework verificable

Re-entrenar el modelo Fashion-MNIST en Lightning + W&B/TensorBoard:

1. LightningModule con train/val/test steps.
2. Trainer con EarlyStopping, ModelCheckpoint, mixed precision.
3. Logging a TensorBoard.
4. Reportar accuracy + screenshot del dashboard.

Criterio de aceptación: accuracy ≥ 0.87; dashboard muestra curvas de train/val.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
self.log no aparece en logger	Falta llamar a self.log('val_loss', loss)
Olvido return loss en training_step	Lightning no puede backprop. Fix: devolver
Multi-GPU con DDP rompe en Jupyter	DDP no funciona en notebooks. Fix: usar dd
Trainer(max_epochs=100) con datasets gigan	Sin max_steps. Fix: max_steps=10_000 como
Mixed precision con OOM	A veces empeora. Fix: bajar batch o usar b

Preguntas frecuentes

Lightning o PyTorch puro?

Lightning para producción / experimentos serios — quita boilerplate, agrega features (distributed, callbacks). Puro para tutoriales y casos muy custom.

Lightning vs HuggingFace Trainer?

Trainer (HF) está más integrado con transformers. Lightning es más general. Si trabajás con LLMs/HF, Trainer; sino Lightning.

FSDP cuándo?

Cuando el modelo no entra en una GPU. Lightning lo activa con strategy='fsdp'.

lightning o pytorch-lightning?

Mismo proyecto. lightning es el nuevo nombre desde 2023.

Logger recomendado?

W&B (Weights & Biases) — mejor DX, comparación de experimentos. TensorBoard si tenés que ser local-only.

Referencias

- Lightning docs.
- Falcon et al. (2019), PyTorch Lightning.
- W&B: <https://wandb.ai/>.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 124 — Clase 124 — tf.data API

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 13 § The Data API. Duración estimada: 65 min.*

Objetivo

Construir pipelines de datos eficientes con tf.data.Dataset: leer desde memoria/archivos/CSV, transformar (map, filter), mezclar (shuffle), batchear (batch), prefetch (paraleliza CPU↔GPU). Saber por qué un buen pipeline de datos es la diferencia entre "GPU al 30 %" y "GPU al 95 %".

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Crear datasets desde varias fuentes: from_tensor_slices, list_files, TextLineDataset.
- Encadenar transformaciones: .map(fn, num_parallel_calls=tf.data.AUTOTUNE), .filter, .shuffle(buffer), .batch(N), .prefetch(tf.data.AUTOTUNE).
- Reconocer el orden correcto: cache → shuffle → batch → prefetch.
- Usar tf.data.AUTOTUNE y profilear con TensorBoard Profiler.
- Saber cuándo cache() vale la pena (datasets que caben en RAM).

Temas

- Lazy evaluation: el dataset es un grafo, no datos cargados.
- shuffle(buffer_size): buffer chico → mal mezclado; buffer = dataset_size → perfecto pero RAM.
- batch(N) → cada elemento es ahora un mini-batch.
- prefetch: solapamiento CPU (loading) con GPU (training).
- interleave para leer múltiples archivos en paralelo.
- Métricas: tf.data.experimental.assert_cardinality.

Definiciones y características

- Dataset.from_tensor_slices: convierte un array NumPy en dataset.
- map(fn, num_parallel_calls=AUTOTUNE): aplica fn a cada elemento; AUTOTUNE elige hilos.
- shuffle(buffer_size): mezclado con buffer rotativo.
- batch(N, drop_remainder=False): agrupa en batches.
- prefetch(n): precarga n batches mientras la GPU procesa el actual.
- cache(): guarda en memoria (o archivo) después de la primera época.
- AUTOTUNE: TF mide y ajusta el número de hilos automáticamente.

Dataset / recursos

- Fashion-MNIST cargado vía tf.data.
- CSV grande para testear pipelines a archivo (anticipa 110 TFRecord).
- Librerías: tensorflow.

Ejercicios

1. Dataset desde NumPy: ds = tf.data.Dataset.from_tensor_slices((x_train, y_train)).shuffle(1024).batch(32).prefetch(tf.data.AUTOTUNE). Iterar y verificar shapes.
2. Map con normalización: .map(lambda x, y: (tf.cast(x, tf.float32)/255., y), num_parallel_calls=tf.data.AUTOTUNE).
3. Cache: comparar tiempo del 1er epoch vs 2do epoch con y sin .cache().
4. Buffer chico: comparar shuffle con buffer_size=10 vs buffer_size=len(data). Inspeccionar el primer batch.

5. Profilear: usar tf.profiler (vía TensorBoard) y verificar dónde está el bottleneck — data loading vs compute.

Homework verificable

Pipeline production-ready para Fashion-MNIST:

1. from_tensor_slices con shuffle, augmentation simple (random flip), normalización, batch=128, prefetch.
2. Cachear el dataset (entra en RAM).
3. Train un MLP por 10 épocas; medir tiempo total.
4. Comparar contra pasar x, y directo a model.fit.

Criterio de aceptación: el pipeline tf.data debe igualar o ser un poco más rápido que pasar arrays directos; con cache, el 2do epoch en adelante es notablemente más rápido (skip de I/O y map).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
shuffle().batch() muy mal mezclado	Buffer = 1000 con dataset de 60 000 → veci
Cache + shuffle en orden incorrecto	.cache().shuffle(...): ok. .shuffle(...).c
.batch(32, drop_remainder=False) y el últi	A veces deseado, a veces rompe modelos cus
GPU idle 70 %	El pipeline es el bottleneck. Fix: prefetc
.map(py_function(...)) lento	Las py_function no se vectorizan ni se eje

Preguntas frecuentes

¿Cuál es el orden canónico de las ops?

Dataset → map → cache → shuffle → batch → prefetch. Cache después de map (para no remap), antes de shuffle (cache de elementos individuales).

¿prefetch(AUTOTUNE) vs prefetch(N)?

AUTOTUNE casi siempre. Mide y ajusta. Solo fijá N si tenés constraints muy específicos.

¿batch(32) o batch(128)?

Depende del modelo y la GPU. Imágenes en VRAM grande: 128. Modelos grandes/Transformers: 8-32. Probá y mirá VRAM.

¿tf.data o PyTorch DataLoader?

Si usás TF: tf.data. PyTorch: DataLoader (similar concepto, num_workers). En 2026, los pipelines de Hugging Face usan datasets library que tiene buen interfaz para ambos.

¿Pipeline en GPU?

Algunas ops sí (decode JPG en GPU con nvidia.dali o tf.image). En general el pipeline corre en CPU y el modelo en GPU; prefetch solapa.

Referencias

- Géron, cap. 13 — The Data API.
- TF — tf.data: Build TensorFlow input pipelines.
- TF — Better performance with the tf.data API.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 125 — Clase 125 — TFRecord

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 13 § The TFRecord Format. Duración estimada: 60 min.*

Objetivo

Aprender el formato TFRecord — el formato binario nativo de TF, optimizado para datasets grandes (cientos de GB) que no caben en RAM. Saber escribirlo (tf.io.TFRecordWriter), parsearlo (tf.io.parse_single_example), y por qué es estándar en TPU/Vertex AI para training a escala.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Serializar un tf.train.Example con Features ↔ Feature (BytesList, Int64List, FloatList).
- Escribir TFRecord shards: with tf.io.TFRecordWriter('file.tfrecord') as w: w.write(serialized).
- Leer con tf.data.TFRecordDataset('file.tfrecord').map(parse_fn).
- Splitear datasets grandes en múltiples shards (*.tfrecord-00000-of-00010) para paralelizar reads.
- Reconocer cuándo TFRecord vale la pena vs alternativas modernas (Parquet, WebDataset).

Temas

- tf.train.Example: estructura protobuf con Features (dict de strings a Feature).
- Feature types: BytesList, Int64List, FloatList.
- Serialize → escribir → leer → parse.
- Sharding: data.tfrecord-NNNNN-of-MMMMM.
- tf.data.experimental.bucket_by_sequence_length para batches eficientes por longitud (NLP).

Definiciones y características

- TFRecord: archivo binario de records seriales protobuf. Eficiente, splitteable, compresible.
- tf.train.Example: proto que envuelve un dict de features.
- SerializeToString(): convierte el Example en bytes para escribir.
- parse_single_example(serialized, feature_spec): parsea con un schema declarado.
- Sharding: dividir un dataset en N archivos → reads paralelos vía interleave.

Dataset / recursos

- Fashion-MNIST exportado a TFRecord.
- Librerías: tensorflow.

Ejercicios

1. Escribir: convertir Fashion-MNIST (60 000 imágenes) a 10 shards TFRecord. Cada Example tiene image (bytes) y label (int).
2. Leer y parsear: ds = tf.data.TFRecordDataset(glob.glob('shards/*.tfrecord')).map(parse_fn). Iterar y verificar shapes.

3. Compresión: escribir con options=tf.io.TFRecordOptions(compression_type='GZIP'). Comparar tamaño.
4. Reads paralelos: ds = ds.interleave(lambda f: tf.data.TFRecordDataset(f), num_parallel_calls=AUTOTUNE). Medir speedup vs lectura serial.
5. Schema: usar tf.io.FixedLenFeature (tamaño fijo) vs VarLenFeature (variable, returns SparseTensor).

Homework verificable

Pipeline completo de Fashion-MNIST con TFRecord:

1. Escribir 10 shards.
2. Leer con interleave + map + batch + prefetch.
3. Entrenar un modelo y comparar tiempo vs cargar desde NumPy.

Criterio de aceptación: el pipeline TFRecord debe escalar mejor cuando los datos no caben en RAM (simular limitando RAM si es necesario); para Fashion-MNIST en RAM, los tiempos deben ser similares.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
parse_single_example falla con "Key not fo	Schema no coincide con lo escrito. Fix: im
Tamaño TFRecord enorme	No comprimiste. Fix: compression_type='GZI
tf.io.parse_tensor vs parse_single_example	Para tensores serializados con tf.io.seria
Reads serializados secuenciales en un únic	El throughput está limitado por 1 disco. F
Sin drop_remainder los últimos batches tie	Algunos modelos rompen. Fix: batch(N, drop

Preguntas frecuentes

¿TFRecord o Parquet?

TFRecord para TF/JAX. Parquet para data science general (Polars, Spark, DuckDB). PyTorch tiene WebDataset que es similar a TFRecord pero más portable (tarballs).

¿Cuándo TFRecord es necesario?

Cuando los datos no caben en RAM (>10 GB) y querés training rápido. Para datasets chicos (< 1 GB), arrays NumPy alcanzan.

¿Cuántos shards?

Regla práctica: 100-1000 archivos, cada uno 100 MB - 1 GB. Para TPU training, ≥ 8 shards por nodo.

¿Comprimir o no?

Comprimir si I/O es el bottleneck. Si CPU es el cuello (decode lento), no comprimir. Probá ambos.

¿Hugging Face datasets?

Sí, formato moderno (basado en Arrow/Parquet) con caching automático y API más amigable. Para LLMs, default industrial. Saber TFRecord es útil legacy pero datasets lo está reemplazando.

Referencias

- Géron, cap. 13 — The TFRecord Format.
- TF — TFRecord and tf.train.Example.
- WebDataset — alternativa PyTorch.

- Hugging Face datasets — alternativa moderna multi-framework.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 126 — Clase 126 — Keras preprocessing layers

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 13 § The Keras Preprocessing Layers. Duración estimada: 65 min.*

Objetivo

Hacer preprocesamiento dentro del modelo con las preprocessing layers de Keras (Normalization, StringLookup, IntegerLookup, Discretization, CategoryEncoding, Hashing, TextVectorization). Beneficio: el preprocesamiento viaja con el modelo (.keras), no como código separado — elimina el clásico "train-serve skew" en producción.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Usar Normalization() con .adapt(data) para escalar features tabulares.
- Usar StringLookup para encoding de categóricas.
- Aplicar TextVectorization para tokenización + indexing.
- Construir un modelo "todo en uno": preprocesamiento + red en el mismo keras.Model.
- Reconocer la ventaja: model.predict(raw_data) funciona, sin necesidad de scaler separado.

Temas

- Normalization: substrae mean, divide por std. .adapt(data) aprende los stats.
- StringLookup / IntegerLookup: mapea categorías a índices enteros.
- CategoryEncoding: one-hot, multi-hot, count.
- Discretization: convierte continua en bucket (bin_boundaries=...).
- Hashing: mapea categorías a buckets via hash (sin necesidad de vocabulario).
- TextVectorization: tokeniza + lookup en una capa.

Definiciones y características

- .adapt(data): aprende los parámetros (mean/std, vocabulario, etc.) desde un dataset. Reemplaza un fit-then-transform separado.
- Normalization: (x - mean) / std. Mean/std aprendidos con .adapt.
- StringLookup(vocabulary=..., output_mode='int'): dict {categoría: índice}. 'multi_hot' para one-hot.
- Hashing(num_bins=1024): para vocabulario gigante o variable. Hash mod num_bins.
- TextVectorization(max_tokens=10_000, output_mode='int', output_sequence_length=100): tokeniza + indexa.

Dataset / recursos

- California Housing (tabular).
- IMDB reviews (texto).

- Librerías: tensorflow, keras.

Ejercicios

1. Normalization tabular: norm = Normalization(); norm.adapt(X_train); X_norm = norm(X_test). Verificar que mean ≈ 0, std ≈ 1.
2. StringLookup: con un array de categorías, lookup = StringLookup(); lookup.adapt(categorias); lookup(['A', 'B', 'C']) → tensor de ints.
3. Modelo end-to-end tabular: inputs = Input((n,)); x = Normalization()(inputs); x = Dense(64, ...)(x); .... Después .adapt(...) la capa norm con X_train.
4. TextVectorization: sobre IMDB, tokenizar, entrenar un modelo de sentimiento.
5. Hashing: con un dataset que tiene 100 000 categorías únicas, Hashing(1024) lo mapea a 1024 buckets. Comparar accuracy vs StringLookup con vocabulario truncado.

Homework verificable

Modelo end-to-end sobre California Housing:

1. Pipeline tf.data con CSV.
2. Modelo con Normalization layer adaptada al train set.
3. Entrenar y guardar con model.save('m.keras').
4. Recargar y predecir sobre datos RAW (sin pre-escalar). Verificar que funciona.

Criterio de aceptación: predicción sobre raw data debe dar el mismo resultado que pre-escalar a mano y pasar al modelo "sin Normalization layer".

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Olvido .adapt() y los stats de Normalizati	Normalization no aprende. Fix: .adapt(trai
.adapt con val/test data	Leakage. Fix: solo train.
StringLookup no encuentra una categoría en	OOV (out-of-vocabulary). Por default se ma
TextVectorization muy lento	Suele ser por max_tokens enorme o output_s
Capa preprocessing fuera del modelo y al s	Anti-pattern. Fix: meter las capas DENTRO

Preguntas frecuentes

¿Normalization o StandardScaler de sklearn?

Si vas a deployar el modelo: Normalization de Keras (viaja con el modelo). Para análisis offline: StandardScaler es igual de bueno y más sklearn-idiomático.

¿Cuándo Hashing vs StringLookup?

Hashing para vocabularios gigantes (>100k) o dinámicos (categorías nuevas aparecen continuamente). StringLookup para vocabulario fijo y manejable.

¿Preprocessing en GPU?

Las preprocessing layers corren en CPU por default. Para mover a GPU: with tf.device('/GPU:0'). La normalización es barata; image augmentation puede ser bueno en GPU.

¿TextVectorization reemplaza a Hugging Face tokenizers?

Para tokenización word-level simple, sí. Para BERT/GPT necesitás los tokenizers específicos (subword/BPE) de Hugging Face — clase 127.

¿Y para LLMs?

LLMs llevan su propio tokenizer (BPE, SentencePiece, tiktoken). TextVectorization no aplica.

Referencias

- Géron, cap. 13 — Preprocessing Data with Keras Preprocessing Layers.
- Keras — Preprocessing layers.
- Keras — Working with preprocessing layers.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 127 — Clase 127 — TensorFlow Datasets (TFDS)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 13 § The TensorFlow Datasets (TFDS) Project. Duración estimada: 40 min.*

Objetivo

Conocer TFDS —catálogo de datasets prearmados (CIFAR, ImageNet, IMDB, COCO, MNIST, GLUE, etc.)— y la alternativa moderna Hugging Face datasets (estándar en NLP/LLMs). Cargar datasets de prueba, hacer splits, y entender por qué TFDS es práctico para benchmarks reproducibles.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Listar datasets disponibles con tfds.list_builders().
- Cargar con tfds.load('cifar10', split=['train', 'test'], as_supervised=True).
- Hacer splits custom con la slicing API: 'train[:80%]', 'train[80%:]'.
- Reconocer cuando usar tfds vs huggingface_hub.datasets.

Temas

- Catálogo TFDS: 200+ datasets, descarga automática + cache.
- as_supervised=True → tuplas (x, y).
- Splits: 'train[:80%]', 'train[-20%:]', 'all'.
- dataset.info con metadata (shape, num_classes, etc.).
- Hugging Face datasets: estándar moderno multi-framework.

Definiciones y características

- TFDS: librería de Google con datasets pre-procesados en formato TFRecord.
- as_supervised: si True, devuelve (input, label). Si False, devuelve dict con todas las features.
- Slicing API: notación tipo Python sobre los splits.
- Hugging Face datasets: equivalente moderno, basado en Arrow, soporta PyTorch/TF/JAX, comunidad enorme.

Dataset / recursos

- CIFAR-10 vía TFDS.

- IMDB vía HF datasets.
- Librerías: tensorflow-datasets (pip install tensorflow-datasets), opcional datasets (HF).

Ejercicios

1. Listar: tfds.list_builders() → primeros 20 datasets.
2. CIFAR-10: (ds_train, ds_test), info = tfds.load('cifar10', split=['train', 'test'], as_supervised=True, with_info=True). Imprimir info.
3. Slicing: cargar train[:90%] + train[90%:] como train/val split.
4. Pipeline: ds_train.map(preprocess).cache().shuffle(1024).batch(32).prefetch(AUTOTUNE).
5. HF datasets: from datasets import load_dataset; ds = load_dataset('imdb'). Inspeccionar; convertir a tf.data con ds.to_tf_dataset(...).

Homework verificable

Entrenar un MLP simple en CIFAR-10 cargado vía TFDS:

1. Cargar con tfds.load(..., as_supervised=True).
2. Pipeline con preprocessing, batch, prefetch.
3. MLP [1024, 512, 256, 10] con BN + Dropout.
4. Reportar accuracy en test.

Criterio de aceptación: accuracy en test ≥ 0.45 (MLP no es ideal para CIFAR — CNN gana — pero debe llegar a 45 %; clase 113+ lo mejora con CNN).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Primera carga muy lenta	TFDS descarga + procesa. Cachea en ~/tenso
OutOfRangeError al iterar	Te quedaste sin batches sin pasar .repeat(
Olvidar as_supervised=True	El dataset devuelve dicts, modelo espera t
tfds.load(split='train+test')	Concatena ambos splits. Para train/val: us
Disco lleno por datasets grandes (ImageNet	TFDS cachea todo. Fix: borrar ~/tensorflow

Preguntas frecuentes

¿TFDS o HF datasets?

Para benchmarks tradicionales (CIFAR, IMDB, MNIST) ambos sirven. Para NLP moderno (text generation, instruction datasets), HF. Para TF nativo + GCP/Vertex AI, TFDS.

¿TFDS funciona con PyTorch?

Sí, con tfds.as_numpy(...) o tfds.load(..., builder_kwargs={'as_dataset_kwargs': ...}). Pero HF es más natural.

¿Datasets propios cargo cómo?

Para datasets locales no listados: tfds.folder_dataset.ImageFolder('/path'), o construir un DatasetBuilder custom. En HF: load_dataset('csv', data_files='...').

¿Datasets de imagen grandes (ImageNet)?

TFDS los maneja bien (shardea, cache). Para TPU + Vertex AI, ya está optimizado.

¿Versionado de datasets?

TFDS versiona por defecto ('cifar10:3.0.2'). HF también. Es importante reportarlo en papers/reports.

Referencias

- Géron, cap. 13 — The TensorFlow Datasets (TFDS) Project.
- TFDS catalog.
- Hugging Face datasets.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 128 — Clase 128 — Capas convolucionales, filtros, feature maps

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 14 § Convolutional Layers. Duración estimada: 75 min.*

Objetivo

Entender la operación de convolución 2D — un filtro K×K se desliza sobre la imagen produciendo un feature map —, los hiperparámetros (filters, kernel_size, strides, padding), por qué las CNN son parameter-efficient vs MLPs (sharing + locality + translation invariance) y cómo aprenden jerarquías visuales (bordes → texturas → partes → objetos).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar Conv2D(filters=32, kernel_size=3, strides=1, padding='same', activation='relu').
- Calcular el shape de la salida: H' = (H - K + 2P)/S + 1.
- Visualizar feature maps (activaciones intermedias) y filtros aprendidos.
- Diferenciar padding='same' (preserva tamaño) de 'valid' (reduce).
- Calcular el # de parámetros de una Conv2D: K K C_in * C_out + C_out (mucho menos que Dense equivalente).

Temas

- Operación convolución 2D paso a paso.
- Filters / kernels como features detectables.
- Feature maps como representación espacial.
- Stride: paso del filtro.
- Padding: bordes; 'same' agrega zeros para preservar shape.
- Parameter sharing: el mismo filtro se aplica en toda la imagen.

Definiciones y características

- Filter / kernel: matriz pequeña (3×3, 5×5) de pesos aprendibles.
- Feature map: salida de aplicar un filtro a toda la imagen. Una capa con 32 filtros produce 32 feature maps.
- Stride: cuántos píxeles se mueve el filtro entre aplicaciones. 1 = denso; 2 = subsampling.
- padding='same': agrega zeros para que la salida tenga la misma altura/ancho que la entrada.
- padding='valid': sin padding. Salida más chica.
- Receptive field: porción de la imagen original que afecta una neurona específica del feature map.

Dataset / recursos

- MNIST / Fashion-MNIST (1 canal) o CIFAR-10 (3 canales).
- Librerías: tensorflow, keras, matplotlib.

Ejercicios

1. Conv básica: Conv2D(8, 3, padding='same', activation='relu')(input_28x28x1). Verificar shape de salida: (batch, 28, 28, 8).
2. Conteo parámetros: para Conv2D(32, kernel_size=5) aplicado a entrada (28, 28, 1), calcular params (551*32 + 32 = 832).
3. Stride 2: Conv2D(32, 3, strides=2, padding='same')(x). Shape de salida: (batch, H/2, W/2, 32).
4. Visualizar feature maps: entrenar una mini-CNN en MNIST; tomar las activaciones intermedias para una imagen y visualizar.
5. Filtros aprendidos: visualizar model.layers[0].kernel.numpy() para los primeros filtros. Para CIFAR, suelen verse bordes/colores básicos.

Homework verificable

Mini-CNN para Fashion-MNIST:

1. Conv(32, 3) → MaxPool(2) → Conv(64, 3) → MaxPool(2) → Flatten → Dense(128) → Dense(10).
2. Entrenar y reportar accuracy.
3. Visualizar feature maps de la primera Conv para 3 imágenes test.
4. Calcular # parámetros total y compararlo con un MLP equivalente en tamaño (Dense(128) → Dense(64) → Dense(10) sin Conv).

Criterio de aceptación: accuracy de la CNN ≥ 0.91 (mejor que MLP); # parámetros de la CNN MLP equivalente.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Input shape error expected ndim=4, got ndi	Olvidaste el canal: MNIST viene (60000, 28
Conv2D + Dense sin Flatten o GlobalAverage	Dense espera 2D. Fix: agregá Flatten() o G
padding='valid' con muchas capas → shape s	Cada conv reduce. Fix: usar 'same' por def
Stride alto pierde información	strides=4 con kernel=3 salta píxeles. Fix:
Filtros aprendidos no se ven nada	Inicialización mala o modelo no entrenado.

Preguntas frecuentes

¿Cuántos filtros?

Empieza chico y duplicá al profundizar. Patrón estándar: 32 → 64 → 128 → 256.

¿Kernel 3×3 o 5×5?

3×3 es default moderno (VGG, ResNet). Dos 3×3 apilados tienen el mismo receptive field que 5×5 con menos parámetros.

¿stride o MaxPool para downsampling?

Tradicionalmente MaxPool. ResNet moderno usa stride 2 en Convs y elimina pooling intermedio. Ambos funcionan.

¿Conv1D para series, Conv3D para video?

Sí. Conv1D para secuencias temporales (clase 121 WaveNet). Conv3D para video (T × H × W × C).

¿Convs vs Transformers en visión?

CNN domina en visión clásica (eficiente, buen prior espacial). ViT (Vision Transformer) supera con datasets muy grandes (>10M imágenes). ConvNeXt (2022) recuperó terreno para CNN. Ver clase 126.

Referencias

- Géron, cap. 14 — Deep Computer Vision Using Convolutional Neural Networks.
- LeCun et al. (1998), Gradient-based learning applied to document recognition — LeNet, paper canónico.
- Keras Conv2D.
- CS231n CNN notes — referencia clásica de Stanford.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 129 — Clase 129 — Pooling

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 14 § Pooling Layers. Duración estimada: 45 min.*

Objetivo

Conocer pooling — operación sin parámetros que reduce dimensiones espaciales: MaxPooling2D, AveragePooling2D, GlobalAveragePooling2D. Saber que max-pool agrega invariancia local a translación y reduce cómputo de capas posteriores. Comparar contra el approach moderno (stride > 1 en Conv).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar MaxPooling2D(2), AveragePooling2D(2), GlobalAveragePooling2D() y conocer sus shapes de salida.
- Diferenciar max-pool (preserva la característica más fuerte) de average-pool (promedio espacial).
- Aplicar GlobalAveragePooling2D antes de la cabeza Dense (estándar en CNN modernas — reemplaza Flatten).
- Reconocer que pooling no tiene parámetros entrenables (es solo una reducción).
- Saber que ResNet/ConvNeXt usan stride en lugar de pool intermedio.

Temas

- MaxPool: ventana k×k → toma el máximo. Default pool_size=2, strides=2 → halve H y W.
- AvgPool: promedio de la ventana. Suaviza.
- GlobalAvgPool: una sola activación por feature map → (batch, channels).
- Invariancia a translación local: si el feature se mueve 1 px, el max no cambia.
- Pool vs stride: equivalentes en muchos casos; tendencia moderna es eliminar pool intermedio.

Definiciones y características

- MaxPooling2D(pool_size=2, strides=2, padding='valid'): ventana 2×2 → 1 valor (el máximo).
- AveragePooling2D: como max pero con promedio.

- GlobalAveragePooling2D: agrega sobre toda la spatial dim → (batch, channels). Sin pool size.
- GlobalMaxPooling2D: similar, con max.
- Output shape: para pool_size=2 stride=2: (H/2, W/2, C).

Dataset / recursos

- Fashion-MNIST o CIFAR-10.
- Librerías: tensorflow, keras.

Ejercicios

1. MaxPool shape: aplicar MaxPool(2) a tensor (1, 28, 28, 32). Verificar salida (1, 14, 14, 32).
2. MaxPool vs AvgPool: con la misma CNN, intercambiar y comparar accuracy en Fashion-MNIST. MaxPool suele ganar marginal.
3. GlobalAvgPool reemplaza Flatten: arquitectura Conv → Conv → GlobalAvgPool → Dense(10) vs Conv → Conv → Flatten → Dense(128) → Dense(10). Comparar params y accuracy.
4. Pool vs stride: comparar Conv(stride=1) → Pool(2) vs Conv(stride=2) (sin pool). En modelos chicos son prácticamente equivalentes.
5. padding='same' en pool: comportamiento si H es impar. valid recorta, same agrega zeros.

Homework verificable

Comparar 3 arquitecturas sobre Fashion-MNIST:

1. Conv(32) → MaxPool → Conv(64) → MaxPool → Flatten → Dense(128) → Dense(10).
2. Conv(32) → MaxPool → Conv(64) → MaxPool → GlobalAvgPool → Dense(10).
3. Conv(32, stride=2) → Conv(64, stride=2) → GlobalAvgPool → Dense(10) (sin pool).

Reportar accuracy y # parámetros.

Criterio de aceptación: la opción 2 tiene muchos menos parámetros y accuracy similar; la 3 también es competitiva (validando el approach moderno).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
pool_size=3, strides=1 recorta poco	Posible pero raro. Fix: usual es pool=2, s
MaxPool antes del primer Conv	Pierde detalle al inicio. Fix: pool después
Olvido pool y modelo Dense tiene millones	Flatten de feature maps grandes. Fix: pool
padding='same' en pool con H impar genera	TF agrega un row/col de zeros. Fix: pre-cr
Usar pooling en NLP / Transformers	No tiene sentido. Fix: pool es espacial; e

Preguntas frecuentes

¿MaxPool o AvgPool?

MaxPool por default — captura "el más fuerte" (útil en clasificación). AvgPool en regresión espacial (segmentación final) o cuando el promedio es semánticamente relevante.

¿GlobalAvgPool obligatorio en ResNet?

Sí, es la última capa antes del FC final. Reduce el riesgo de overfit del Flatten + Dense gigante.

¿Pool intermedio aún se usa?

CNNs clásicas (VGG, LeNet) sí. ResNet/ConvNeXt usan stride en conv (sin pool intermedio).

¿pool_size=4 o más grande?

Raro. Casi siempre 2. Más grande pierde información rápido.

¿Pool diferenciable?

MaxPool tiene gradiente subdiferencial (= 1 en el max, 0 en el resto). Avg es diferenciable normal. TF/PyTorch los manejan transparente.

Referencias

- Géron, cap. 14 — Pooling Layers.
- Keras MaxPooling2D.
- Springenberg et al. (2015), Striving for Simplicity: The All Convolutional Net — argumenta eliminar pool intermedio.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 130 — Clase 130 — Arquitecturas CNN: LeNet, AlexNet, VGG, GoogLeNet, ResNet, Xception, SENet, EfficientNet, ConvNeXt

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 14 § CNN Architectures + papers originales. Duración estimada: 95 min.*

Objetivo

Conocer la historia y evolución de las arquitecturas CNN desde LeNet-5 (1998) hasta ConvNeXt (2022) — qué innovación introdujo cada una, por qué importaba, y cuál usar hoy. Identificar 3 patrones clave: profundidad creciente, módulos con paths múltiples, eficiencia paramétrica.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Trazar la línea temporal: LeNet → AlexNet → VGG → GoogLeNet (Inception) → ResNet → Xception → SENet → EfficientNet → ConvNeXt.
- Reconocer qué innovación trajo cada una: ReLU + dropout (AlexNet), kernels 3×3 (VGG), módulos Inception (GoogLeNet), skip connections (ResNet), depthwise-separable (Xception), squeeze-and-excite (SENet), compound scaling (EfficientNet), modernización con receta ConvNeXt.
- Implementar un mini-ResNet con Add() y skip connections.
- Cargar arquitecturas de keras.applications y leer sus tamaños (MobileNetV3, EfficientNetB0, ConvNeXtTiny).
- Elegir la arquitectura adecuada según constraint (latency, accuracy, memory).

Temas

- LeNet-5 (1998): 7 capas, MNIST. La cuna.
- AlexNet (2012): GPU + ReLU + Dropout. Ganó ImageNet → revolución DL.
- VGG (2014): muy uniforme — solo conv 3×3 + maxpool. 138M parámetros.

- GoogLeNet / Inception (2014): módulos paralelos con kernels 1×1, 3×3, 5×5.
- ResNet (2015): skip connections → entrenamiento de redes de 152 capas posible. Cambió el juego.
- Xception (2017): depthwise-separable convolutions → eficiencia.
- SENet (2017): attention sobre canales (squeeze-and-excite).
- EfficientNet (2019): scaling balanceado de depth/width/resolution.
- ConvNeXt (2022): "modernizar ResNet" con trucos de ViT.

Definiciones y características

- Skip / residual connection: y = x + F(x). Permite gradiente fluir y entrenar redes muy profundas.
- Bottleneck: 1×1 conv → 3×3 conv → 1×1 conv (reduce → opera → expande). Reduce parámetros.
- Depthwise-separable conv: una conv que actúa por canal, seguida de 1×1 conv que mezcla canales. ~10× menos parámetros.
- Squeeze-and-Excite: re-pesa los canales aprendido (GlobalAvgPool → Dense → sigmoid → multiply).
- Compound scaling (EfficientNet): escalar depth, width, resolution juntos con coeficientes balanceados.
- keras.applications: catálogo con pesos ImageNet preentrenados de todas estas arquitecturas.

Dataset / recursos

- ImageNet (vía transfer) o un dataset propio chico.
- Librerías: tensorflow, keras.applications.

Ejercicios

1. Cargar varios modelos: keras.applications.{ResNet50, EfficientNetB0, ConvNeXtTiny}(weights='imagenet'). Comparar model.count_params() y model.summary().
2. Mini-ResNet: implementar 4 bloques residuales: def res_block(x): return x + Conv(64,3,padding='same')(ReLU()(Conv(64,3,padding='same')(x))). Apilar y entrenar.
3. Skip connection a mano: comparar entrenamiento de un "ResNet" sin skip vs con skip a 50 capas. Sin skip no entrena.
4. Depthwise-separable: usar SeparableConv2D en lugar de Conv2D en el mismo modelo. Comparar params y accuracy.
5. Squeeze-Excite: implementar un bloque SE manualmente: s = GlobalAvgPool(x); s = Dense(C//r)(s); s = Dense(C, sigmoid)(s); return x * Reshape((1,1,C))(s).

Homework verificable

Comparar 4 arquitecturas en un dataset propio de imágenes (≥ 5 clases) con transfer learning:

1. ResNet50 (clásico).
2. EfficientNetB0 (eficiente).
3. MobileNetV3Small (móvil).
4. ConvNeXtTiny (moderno).

Para cada uno: feature extraction (frozen) + fine-tune. Reportar accuracy, # parámetros, tiempo de inference.

Criterio de aceptación: ConvNeXt o EfficientNet deben ganar en accuracy; MobileNet en velocidad de inference. ResNet sirve de baseline.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Cargar ResNet50 sin include_top=False y pa	El head ImageNet espera 1000 clases. Fix:
Cargar EfficientNet y pasarle imágenes en	Espera preprocesamiento específico. Fix: k

Implementar ResNet desde cero y Add() fall	Skip connection requiere shapes iguales —
MobileNet para batch grande en CPU lento	Está optimizado para GPU/mobile. Fix: para
ConvNeXt en TF Lite no exporta	Algunos ops de ConvNeXt aún no portados a

Preguntas frecuentes

¿Cuál arquitectura uso en 2026?

- Default: EfficientNetB0 o ConvNeXtTiny.
- Movil/embedded: MobileNetV3Small.
- Máxima accuracy con budget: ConvNeXt-L o EfficientNet-L2.
- Visión + texto unificado: CLIP / SigLIP (clase 129).

¿ResNet aún se usa?

Sí, como baseline universal. Confiable, bien entendida, todo framework la soporta.

¿ViT supera a CNNs?

Con datasets muy grandes (>10M imágenes), ViT (clase 126) gana. Con datasets típicos del cliente (<100k imágenes), CNNs ganan o empatan. ConvNeXt mostró que CNNs bien modernizadas igualan a ViT.

¿"Compound scaling" qué significa?

EfficientNet showed que escalar depth × width × resolution con coeficientes balanceados (φ) es más eficiente que escalar uno solo. Por eso B0, B1, B2... son la misma red escalada.

¿Tantas opciones por qué?

Cada una optimizó una métrica distinta (accuracy puro, params, latencia, eficiencia mobile). No hay "una mejor" — depende del constraint.

Referencias

- Géron, cap. 14 — CNN Architectures.
- Krizhevsky et al. (2012), AlexNet, NeurIPS.
- Simonyan & Zisserman (2014), VGG.
- Szegedy et al. (2015), Inception (GoogLeNet), CVPR.
- He et al. (2015), Deep Residual Learning (ResNet), CVPR.
- Chollet (2017), Xception.
- Hu et al. (2018), Squeeze-and-Excitation Networks, CVPR.
- Tan & Le (2019), EfficientNet, ICML.
- Liu et al. (2022), A ConvNet for the 2020s (ConvNeXt), CVPR.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 131 — Clase 131 — Transfer learning con CNNs preentrenadas

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 14 § Using Pretrained Models from Keras. Duración estimada: 70 min.*

Objetivo

Aplicar transfer learning específicamente en visión — el caso de uso más común del campo. Profundizar la clase 101 con receta industrial: data augmentation, fine-tuning gradual, learning rate diferencial, y manejo de BatchNorm en fine-tuning (un gotcha clásico).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Construir pipeline con image_dataset_from_directory + augmentation (RandomFlip, RandomRotation, RandomZoom).
- Aplicar el preprocess_input específico del modelo base.
- Hacer fine-tuning en 2 fases con LR diferencial.
- Manejar correctamente BatchNorm en fine-tuning (mantenerlo en inference mode si descongelaste pocas capas).
- Comparar feature extraction (frozen base + nueva head) vs fine-tune (descongelar todo) según tamaño del dataset.

Temas

- Augmentation como capa: RandomFlip, RandomRotation, RandomZoom, RandomCrop, RandomContrast.
- preprocess_input por modelo (cada red espera su escalado).
- 2-stage training: freeze + head warmup → unfreeze + LR bajo.
- BN gotcha: cuando descongelás, BN sigue actualizando moving averages → puede dañar pretraining.
- MixUp, CutMix, RandAugment como augmentations modernas (anticipo).

Definiciones y características

- Augmentation layer: capa Keras que transforma random las imágenes durante training, no en inference.
- preprocess_input: función específica del modelo (keras.applications.<modelo>.preprocess_input).
- Feature extraction: base frozen, solo entreno head. Bueno para n < 1000.
- Fine-tune: descongelar parte/total y reentrenar con LR muy bajo.
- LR diferencial: capas tempranas LR bajo (1e-5), tardías más alto (1e-4).

Dataset / recursos

- Dataset chico propio o tfds.load('cats_vs_dogs') o tfds.load('tf_flowers').
- Modelo base: EfficientNetB0 o MobileNetV3Small.
- Librerías: tensorflow, keras, keras.applications.

Ejercicios

1. Pipeline con augmentation: RandomFlip + RandomRotation(0.1) + RandomZoom(0.1) como capas. Visualizar imágenes augmentadas.
2. Modelo con base + head: base = EfficientNetB0(include_top=False, weights='imagenet'); base.trainable = False; model = Sequential([data_aug, preprocess_input, base, GlobalAvgPool, Dropout, Dense(num_classes)]).
3. Etapa 1 (head warmup): Adam(1e-3), entrenar 5 épocas.
4. Etapa 2 (fine-tune): base.trainable = True, recompilar con Adam(1e-5). Entrenar 10 épocas. Verificar mejora.
5. BN gotcha: con base.trainable = True pero pasando training=False a la base durante fine-tuning → BN no se actualiza. Comparar.

Homework verificable

Sobre un dataset de 5 categorías (e.g., flores):

1. Pipeline completo con augmentation.
2. Transfer learning con EfficientNetB0 en 2 fases.
3. Reportar accuracy de cada fase.
4. Comparar contra entrenar EfficientNet desde cero (sin transfer).

Criterio de aceptación: la etapa 2 mejora la 1 en ≥ 2 pp; transfer learning supera "desde cero" por mucho margen con dataset chico.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Imágenes pasadas en [0, 255] cuando el mod	Fix: agregar preprocess_input correcto.
Augmentation aplicada en val/test también	Augmentation debe ser solo en train. Fix:
Fine-tuning con LR=1e-3 → catastrophic for	Fix: LR 10-100× más bajo.
Forgot to recompile después de cambiar tra	El optimizer no ve los nuevos pesos como t
BatchNorm en fine-tuning actualiza moving	Para datasets chicos, mantener BN en infer

Preguntas frecuentes

¿Cuántas imágenes mínimas?

50-100 por clase para feature extraction. 200-500 por clase para fine-tuning completo. < 50 → considerar data augmentation agresiva o few-shot learning.

¿Qué proporción de capas descongelo?

Empezá con todo. Si overfittea, congelar las primeras N. Si underfittea, descongelar todo.

¿include_top=False siempre?

Sí para transfer learning (querés head custom).

¿Augmentation en GPU o CPU?

GPU es más rápido si está disponible (with tf.device('/GPU:0')). Augmentation pesada es lo más comúnmente bottleneck en CPU.

¿Test-time augmentation (TTA)?

Predecir varias versiones augmentadas de la misma imagen y promediar. Mejora accuracy 0.5-2 pp. Buen truco para Kaggle.

Referencias

- Géron, cap. 14 — Using Pretrained Models from Keras.
- Yosinski et al. (2014), How transferable are features in deep neural networks?, NeurIPS.
- Cubuk et al. (2020), RandAugment.
- Keras — Image data preprocessing.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.

- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 132 — Clase 132 — Localización, detección, segmentación (+ DETR, Segment Anything, YOLOv11)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 14 § Object Detection y § Semantic Segmentation + papers DETR, SAM, YOLOv11. Duración estimada: 90 min.*

Objetivo

Saber detectar y segmentar objetos en imágenes — la tarea de visión más compleja y más comercial. Conocer la evolución: Faster R-CNN (two-stage, lento + preciso) → YOLO (one-stage, rápido) → DETR (Transformer, end-to-end) → YOLOv11 (estado del arte 2024) → Segment Anything (SAM/SAM 2) (foundation model para segmentación).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Distinguir las 4 tareas: clasificación, localización (1 objeto), detección (N objetos + cajas), segmentación (pixel-wise mask).
- Usar un modelo YOLOv11 preentrenado con ultralytics: model = YOLO('yolo11n.pt'); results = model('imagen.jpg').
- Aplicar Segment Anything (SAM 2) para segmentación promptable con puntos o cajas como input.
- Reconocer cuándo elegir DETR (mejor formal, lento) vs YOLO (rápido, default industrial) vs SAM (pre-entrenado universal).
- Métricas: IoU, mAP, COCO AP@[.5:.05:.95].

Temas

- Localización vs detección vs instance segmentation vs semantic segmentation.
- IoU, mAP, COCO benchmark.
- Anchors vs anchor-free.
- One-stage (YOLO, SSD) vs two-stage (Faster R-CNN).
- Complemento moderno: DETR (Carion et al. 2020), SAM/SAM 2 (Meta 2023/2024), YOLOv11 (Ultralytics 2024).
- Pre-trained pipelines: ultralytics, transformers (DETR), segment_anything.

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 117b — Segment Anything (SAM / SAM 2)
- Clase 117c — YOLOv11 práctico: detección, segmentación, pose, tracking

Definiciones y características

- Bounding box: (x_min, y_min, x_max, y_max) o (cx, cy, w, h).
- IoU (Intersection over Union): area(intersección) / area(unión). Match si IoU > 0.5.
- mAP (mean Average Precision): promedio del AP sobre clases y umbrales IoU. COCO usa IoU 0.5 → 0.95 en pasos de 0.05.

- NMS (Non-Maximum Suppression): post-procesado que elimina cajas superpuestas dejando la de mejor score. DETR no la necesita.
- Anchor-based: pre-define cajas en cada posición y aprende offsets.
- Anchor-free: predice directamente, sin pre-definir cajas (FCOS, DETR, YOLOX+).
- Semantic vs instance segmentation: semantic = mismo color para todas las instancias de "persona"; instance = cada persona con su color.
- Promptable segmentation (SAM): dado un punto/caja/texto, devolver la máscara correspondiente.

Dataset / recursos

- COCO (detección/seg estándar): tfds.load('coco/2017').
- Pascal VOC: más chico, bueno para experimentos.
- Custom: anotaciones en YOLO format (txt por imagen) o COCO format (JSON).
- Librerías: ultralytics (pip install ultralytics), transformers, segment-anything.

Ejercicios

1. YOLO inference: cargar yolo11n.pt y detectar sobre 3 imágenes propias. Visualizar boxes + labels.
2. DETR inference: usar facebook/detr-resnet-50 desde HF. Comparar resultados con YOLO sobre las mismas imágenes.
3. SAM segmentation: dar un punto sobre un objeto en una imagen, obtener máscara. Probar con cajas.
4. mAP a mano: dado un set de predicciones y GT, implementar IoU y calcular AP@0.5 manualmente.
5. YOLO fine-tune: dataset propio (≥ 100 imágenes anotadas), model.train(data='dataset.yaml', epochs=50). Reportar mAP.

Homework verificable

Detector custom de 2-3 clases con YOLOv11:

1. Etiquetar ~100 imágenes con CVAT, LabelImg o Roboflow → YOLO format.
2. dataset.yaml con paths + clases.
3. model.train(data='dataset.yaml', epochs=100, imgsz=640).
4. Reportar mAP@0.5 en val + 3 imágenes con bounding boxes superpuestos.

Criterio de aceptación: mAP@0.5 ≥ 0.7 sobre val set; el modelo detecta correctamente las clases en las 3 imágenes de inspección visual.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
YOLO devuelve cajas raras tras fine-tuning	Pocos datos o etiquetado inconsistente. Fi
imgsz=320 para mejor velocidad pierde much	Trade-off speed/accuracy. Fix: 640 default
SAM lento sin GPU	El modelo ViT-H tiene 636M params. Fix: vi
DETR lento en training	DETR converge muy lento (500 epochs en COC
NMS no aplicado → muchas cajas superpuesta	YOLO lo aplica internamente (conf=0.25, io

Preguntas frecuentes

¿YOLO o DETR para producción?

YOLO es default industrial (rápido, bien soportado, fácil deploy). DETR es preferido en investigación.

¿SAM puede reemplazar a un segmentador clásico?

Para tareas donde quieras "segmentar lo que el usuario apunta", sí. Para clases específicas sin intervención

humana, fine-tunear un segmentador es más eficiente.

¿Open-vocabulary detection?

Detectores que aceptan clases nuevas vía texto sin reentrenar (CLIP-based). OWL-ViT, GroundingDINO. Útil cuando las clases cambian.

¿Anotación manual cuántas imágenes?

Para 2-3 clases simples con YOLO: 100-500 ya da resultados decentes. Para escenarios complejos: miles. Roboflow / CVAT facilitan el workflow.

¿Tracking de objetos en video?

YOLO + tracker (ByteTrack, BoT-SORT) integrados en ultralytics. Para video más complejo, SAM 2.

Referencias

- Géron, cap. 14 — Object Detection y Semantic Segmentation.
- Carion et al. (2020), End-to-End Object Detection with Transformers (DETR), ECCV.
- Kirillov et al. (2023), Segment Anything, ICCV.
- Ravi et al. (2024), SAM 2: Segment Anything in Images and Videos, Meta.
- Ultralytics YOLOv11 docs.
- Segment Anything project.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 133 — Clase 133 — Segment Anything (SAM / SAM 2): foundation model para segmentación

> *Parte: 2 — Deep Learning · Fuente: Kirillov et al. (2023) + Ravi et al. (2024) SAM 2. Duración estimada: 85 min.*

Objetivo

Usar Segment Anything (Meta AI 2023) y SAM 2 (2024) — el foundation model para segmentación: entrenado en 11M imágenes + 1.1B máscaras (SAM 2 agrega video). Segmenta cualquier objeto dado un prompt (punto, caja, máscara, "todo"). Zero-shot — no requiere training para empezar.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Instalar y cargar SAM/SAM 2 con pip install 'git+https://github.com/facebookresearch/segment-anything-2'.
- Aplicar prompts: punto, caja, multi-punto positivo/negativo.
- Generar máscaras en modo "everything" (segmenta cada objeto automáticamente).
- Tracking de máscaras en video con SAM 2 (memoria temporal).
- Combinar SAM con detector (YOLO) para pipeline detection → segmentation.

Temas

- Arquitectura SAM: ViT encoder + prompt encoder + mask decoder.
- Promptable: una sola red, múltiples interfaces.
- SAM 2: adds memory + tracking en video.
- Variants: vit_h (mejor calidad), vit_l, vit_b (más rápido).
- Pipeline detección + segmentación: YOLO/Grounding DINO → boxes → SAM → masks.
- Fine-tuning SAM (rara vez necesario, casos médicos / dominios muy específicos).

Definiciones y características

- Promptable segmentation: input = imagen + prompt (punto/caja/mask) → output = mask.
- Mask decoder: producirá hasta 3 máscaras por prompt (handles ambigüedad).
- SamPredictor: API estándar para una imagen.
- SamAutomaticMaskGenerator: modo "everything" — segmenta toda la imagen automáticamente.
- SAM 2 memory: bank de features previas para tracking en video.

Dataset / recursos

- Imágenes propias o cualquier dataset visual.
- Modelos pretrained: <https://github.com/facebookresearch/segment-anything-2>.
- Librerías: segment-anything-2 (PyTorch).

Ejercicios

1. SAM setup: descargar checkpoint vit_h. Cargar con SamPredictor.
2. Punto prompt: predictor.set_image(img); masks, scores, _ = predictor.predict(point_coords=[[x,y]], point_labels=[1]).
3. Box prompt: pasar box=[x1,y1,x2,y2]; útil después de YOLO.
4. Everything mode: SamAutomaticMaskGenerator(sam).generate(img) → lista de masks.
5. SAM 2 video tracking: tomar un punto en frame 0, propagar a través del video.

Homework verificable

Pipeline YOLO + SAM para anotación semi-automática:

1. YOLOv11 detecta objects → boxes.
2. Para cada box, SAM produce máscara precisa.
3. Visualizar overlay sobre imagen.
4. Reportar tiempo por imagen.

Criterio de aceptación: las máscaras son visualmente correctas (alineadas a contornos); pipeline corre en GPU < 1s por imagen.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
SAM lento en CPU	Es enorme. Fix: usar vit_b o GPU.
Máscara incluye background	Prompt ambiguo. Fix: agregar puntos negati
Out-of-memory con ViT-H	4-8 GB VRAM necesarios. Fix: vit_l o ViT-B
Mask decoder devuelve 3 masks, ¿cuál uso?	El score más alto suele ser la correcta. F
SAM 2 tracking pierde objeto en video	Re-prompt cada N frames.

Preguntas frecuentes

¿SAM reemplaza segmentación supervisada?

Para uso interactivo / anotación, sí. Para producción con clases específicas sin intervención humana, fine-tune un segmentador clásico (DeepLabV3+, YOLO-seg) es más eficiente.

¿Open vocabulary?

SAM solo: NO. Para "segmenta el perro" con texto: combinar con Grounding DINO (detector text-grounded) → caja → SAM → mask. O usar Grounded-SAM end-to-end.

¿SAM en mobile?

Hay versiones distilled: MobileSAM, EfficientSAM — 10× más rápidas, ligera pérdida de calidad.

¿Fine-tune SAM?

Posible pero rara vez vale la pena. Casos médicos / dominios muy distintos.

¿Licencia?

Apache 2.0. Comercialmente OK.

Referencias

- Kirillov et al. (2023), Segment Anything, ICCV.
- Ravi et al. (2024), SAM 2: Segment Anything in Images and Videos.
- Segment Anything project.
- Grounded-SAM.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 134 — Clase 134 — YOLOv11 práctico: detección, segmentación, pose, tracking

> *Parte: 2 — Deep Learning · Fuente: Ultralytics YOLOv11 docs. Duración estimada: 85 min.*

Objetivo

Dominar YOLOv11 (Ultralytics, 2024) — el detector default industrial 2026: inference real-time, fine-tuning sencillo, export a ONNX/TensorRT/CoreML/TFLite con un kwarg. Cubrir las 4 tareas: detection, segmentation, pose estimation, oriented bounding boxes (OBB).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Inferir con un modelo COCO-pretrained: YOLO('yolo11n.pt')(img).
- Fine-tunear con dataset propio en formato YOLO (txt por imagen).
- Aplicar las 4 tareas: detection (yolo11n.pt), segmentation (yolo11n-seg.pt), pose (yolo11n-pose.pt), OBB (yolo11n-obb.pt).
- Tracking de objetos en video con ByteTrack / BoT-SORT integrado.

- Exportar a ONNX/TensorRT para deploy.

Temas

- Modelos: n (nano), s, m, l, x. Trade-off speed vs accuracy.
- Formato YOLO de anotación: class cx cy w h (normalizado).
- dataset.yaml: paths + nombres de clases.
- Fine-tuning: model.train(data='dataset.yaml', epochs=100, imgsz=640).
- Modes: predict, val, train, export, benchmark.
- Tracking integrado.

Definiciones y características

- YOLO: clase Ultralytics que carga cualquier variante.
- results[0]: objeto con boxes, masks (seg), keypoints (pose), probs.
- OBB (Oriented Bounding Box): caja rotada — útil para aerial, documents.
- model.export(format='onnx'): 1 línea para ONNX/TensorRT/CoreML/TFLite.
- mAP@0.5: métrica standard, threshold IoU 0.5.

Dataset / recursos

- COCO pre-trained para inference.
- Roboflow Universe para dataset propio.
- Librerías: ultralytics (pip install ultralytics).

Ejercicios

1. Inference: model = YOLO('yolo11n.pt'); results = model('zidane.jpg'); results[0].show().
2. Segmentation: YOLO('yolo11n-seg.pt'). Visualizar máscaras.
3. Pose: YOLO('yolo11n-pose.pt'). Detectar keypoints en una foto.
4. Fine-tune: dataset propio (50-100 imágenes anotadas). model.train(data='ds.yaml', epochs=50, imgsz=640).
5. Tracking: model.track('video.mp4', tracker='botsort.yaml').

Homework verificable

Detector custom de 2 clases:

1. Etiquetar ~100 imágenes con Roboflow / LabelImg → formato YOLO.
2. dataset.yaml correcto.
3. Train YOLOv11n por 100 épocas.
4. Reportar mAP@0.5; visualizar 3 inferencias.
5. Exportar a ONNX y verificar que predice igual.

Criterio de aceptación: mAP@0.5 ≥ 0.7; ONNX produce predicciones consistentes.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
dataset.yaml paths mal	Errores de path. Fix: usar paths absolutos
mAP bajo con dataset chico	Pocos datos. Fix: ≥ 100 ejemplos por clase
Inference lento en CPU	YOLO es para GPU. Fix: device='cuda' o exp
Class names mismatched	Orden en yaml debe coincidir con anotacion
Anotaciones con coords absolutas	YOLO espera normalizadas [0,1]. Fix: conve

Preguntas frecuentes

YOLOv11 vs versiones anteriores?

v11 mejor accuracy/latency. v8 sigue siendo válido y muy usado. v5 legacy pero funciona.

Tareas además de detección?

Segmentation, pose estimation (COCO keypoints), classification, OBB. Una sola API.

Producción cómo deploy?

Export → ONNX → ONNX Runtime / TensorRT. O TF Lite para mobile. Soportado nativo.

Licencia?

AGPL para uso open-source. Para uso comercial cerrado, license enterprise.

Augmentation automática?

Sí, mosaic/mixup/etc. Built-in. Tunear con model.train(augment=True).

Referencias

- Ultralytics YOLOv11.
- Redmon et al. (2016), You Only Look Once — paper original.
- Roboflow Universe — datasets pre-anotados.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 135 — Clase 135 — RNNs: neuronas recurrentes, BPTT

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 15 § Recurrent Neurons and Layers. Duración estimada: 70 min.*

Objetivo

Entender las redes recurrentes (RNN) — la primera arquitectura para secuencias: misma celda aplicada en cada timestep, estado oculto h_t que acumula contexto, y BPTT (Backpropagation Through Time) que entrena estos modelos. Reconocer sus limitaciones (vanishing en secuencias largas) que motivaron LSTM (clase 120) y eventualmente Transformers.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar la ecuación h_t = tanh(W_h · h_{t-1} + W_x · x_t + b).
- Usar keras.layers.SimpleRNN(units=N, return_sequences=False|True).
- Diferenciar return_sequences=False (un único output al final) vs True (output por timestep).
- Implementar BPTT truncado (longitud máxima de window).
- Reconocer cuándo una RNN simple es suficiente (secuencias cortas, < 20 pasos) y cuándo necesitás LSTM/Transformer.

Temas

- Estado oculto h_t como memoria.
- Unfolding: la RNN como red feedforward muy profunda en el tiempo.
- BPTT: gradientes a través de cada paso temporal.
- Vanishing/exploding en secuencias largas (compounding multiplicativo).
- BPTT truncado.

Definiciones y características

- RNN cell: función (h_{t-1}, x_t) → h_t.
- return_sequences=True: output (batch, T, units) — para apilar otra RNN o aplicar Dense a cada timestep.
- return_state=True: devuelve también el estado final (útil para encoder-decoder).
- BPTT: backprop estándar aplicado al grafo unfolded.
- Truncated BPTT: cortar el gradiente cada K pasos para evitar costo + vanishing.

Dataset / recursos

- Serie temporal sintética (sumas, sine).
- Librerías: tensorflow, keras, numpy, matplotlib.

Ejercicios

1. SimpleRNN básico: predecir el siguiente valor de sin(t). Modelo: SimpleRNN(20) → Dense(1). Entrenar y graficar predicciones.
2. return_sequences=True: apilar 2 RNN, primera con return_sequences=True, segunda sin. Verificar shapes.
3. Predicción de N pasos adelante: iterar prediciendo, alimentando la predicción anterior como nuevo input.
4. BPTT truncado: con secuencia de 200 pasos, comparar BPTT completo vs truncado a 20.
5. Vanishing demo: usar SimpleRNN con secuencias de 100 pasos. Las primeras observaciones casi no influyen en la última predicción.

Homework verificable

Forecasting con SimpleRNN sobre sin(t) + ruido:

1. Generar 5 000 pasos de sin(0.01·t) + N(0, 0.05).
2. Construir samples de longitud 50 → predecir el paso 51.
3. SimpleRNN [20] + Dense(1).
4. Evaluar MAE en test y graficar predicción vs realidad.

Criterio de aceptación: MAE < 0.1; gráfico muestra la predicción siguiendo la sinusoide.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Input shape error en RNN	RNN espera (batch, timesteps, features). F
Olvido return_sequences=True al apilar	Capa intermedia debe devolver toda la secu
Loss explota con secuencia larga	Exploding gradients. Fix: clipnorm=1.0 + L
Predicción autoregresiva diverge	Error acumulado. Fix: usar teacher forcing
mask_zero no soportado en SimpleRNN	Es para Embedding + LSTM/GRU. Fix: usar LS

Preguntas frecuentes

¿RNN vs LSTM vs Transformer en 2026?

Para secuencias cortas y simples: SimpleRNN o LSTM. Para >50 pasos o NLP serio: Transformer. SimpleRNN ya casi no se usa en producción salvo casos muy específicos.

¿return_state para qué?

Para encoder-decoder: el estado final del encoder inicializa el decoder.

¿Bidireccional?

Bidirectional(SimpleRNN(...)) corre la RNN forward y backward, concatenando. Mejor para tareas non-causales (clasificación, NER).

¿Cuántas units?

Empezá con 32-128. Más complejidad de secuencia → más units. RNNs no son tan parameter-hungry como Transformers.

¿Predecir 1 paso o varios?

Modelos de "1 paso" suelen ser más precisos. Para "varios", entrenar con return_sequences=True y target shifted, o seq2seq (clase 124).

Referencias

- Géron, cap. 15 — Recurrent Neural Networks.
- Pascanu et al. (2013), On the difficulty of training RNNs.
- Keras RNN guide.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 136 — Clase 136 — Forecasting de series con RNN

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 15 § Forecasting a Time Series. Duración estimada: 80 min.*

Objetivo

Aplicar RNN/LSTM/GRU a un problema real de forecasting de series temporales. Hacer split temporal (NO aleatorio), preparar windows, comparar contra baselines (naive, MA, ARIMA), y reportar métricas estándar (MAE, MAPE, RMSE).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Hacer split temporal (train: período antiguo, val/test: período más reciente).
- Construir samples de longitud T con tf.keras.utils.timeseries_dataset_from_array.
- Comparar contra el baseline naïve (ŷ_t = y_{t-1}) y media móvil.

- Reportar MAE, MAPE, RMSE.
- Reconocer cuándo Deep Learning es mejor que ARIMA / Prophet / XGBoost para series.

Temas

- Split temporal estricto (no shuffle).
- Stationarity / diferenciación.
- Baseline naïve y por qué siempre comparar contra él.
- Windowing: cómo elegir tamaño de window (T) y horizonte.
- Forecasting multi-step: directo vs recursivo vs seq2seq.

Definiciones y características

- Window / lookback: cuántos pasos pasados usás para predecir.
- Horizon: cuántos pasos hacia el futuro predecís.
- Naïve forecast: predicción = último valor observado.
- MAPE: mean(|y - ŷ| / |y|) × 100 — error porcentual.
- timeseries_dataset_from_array: helper Keras para crear ds de windows automáticamente.

Dataset / recursos

- seaborn flights dataset o cualquier serie pública (e.g., consumo eléctrico).
- tfds.load('electricity_load_diagrams') o sintético.
- Librerías: tensorflow, keras, pandas, matplotlib.

Ejercicios

1. Split temporal: separar primer 70 % train, 15 % val, último 15 % test. No mezclar.
2. Baseline naïve: y_pred = y_test.shift(1). Reportar MAE.
3. LSTM forecasting: Sequential([LSTM(32), Dense(1)]). Comparar contra naïve.
4. GRU: igual con GRU. Comparar performance y velocidad.
5. Multi-step: predecir 7 pasos directamente (Dense(7) al final). Comparar contra predicción recursiva.

Homework verificable

Forecasting de consumo eléctrico:

1. Dataset diario, 5 años; split 70/15/15 temporal.
2. Baselines: naïve y MA(7).
3. LSTM [64], lookback=30 días, horizon=7 días.
4. Reportar MAE en test para los 3.

Criterio de aceptación: LSTM debe superar a naive en MAE; comparable o mejor que MA(7). Documentar dónde gana y dónde pierde.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
train_test_split con shuffle por costumbre	Filtración de futuro al pasado. Fix: split
LSTM mejor que naïve por 0.001 → declarar	Sin significancia clara. Fix: usar pingoui
Normalizar usando stats de todo el dataset	Leakage. Fix: fitear solo en train.
Predicción multi-step recursiva diverge	Error acumulado. Fix: entrenar con teacher
Reportar MAPE cuando hay valores cerca de	MAPE explota. Fix: usar SMAPE o reportar M

Preguntas frecuentes

¿DL siempre supera a ARIMA?

No. Para series cortas, regulares, sin features exógenas: ARIMA / ETS suelen igualar o ganar. DL gana con series largas + features adicionales + complejidad no lineal.

¿Lookback óptimo?

Empezar con 1-2 ciclos (e.g., 14 días si hay estacionalidad semanal). Tunear con val.

¿RNN o Transformer para series?

Transformers de series temporales (TFT, Informer, PatchTST 2023) son state-of-the-art para series largas con muchas features. Para series chicas, LSTM/GRU + features manuales gana.

¿Multi-step directo o recursivo?

Directo: 1 modelo predice todos los horizontes. Recursivo: feedback de cada paso. Directo más simple y a veces mejor en horizonte largo.

¿Cómo manejo estacionalidad?

Features explícitas (mes, día de semana, hora) ayudan mucho. Diferenciar la serie también.

Referencias

- Géron, cap. 15 — Forecasting a Time Series.
- Hyndman & Athanasopoulos (2021), Forecasting: Principles and Practice (libro gratuito).
- Lim et al. (2021), Temporal Fusion Transformers, IJF.
- Nie et al. (2023), PatchTST, ICLR.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 137 — Clase 137 — LSTM, GRU

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 15 § Tackling Short-Term Memory Problems. Duración estimada: 70 min.*

Objetivo

Entender LSTM (Hochreiter & Schmidhuber 1997) y GRU (Cho et al. 2014) — celdas recurrentes con gates que solucionan el vanishing gradient de SimpleRNN: la información puede fluir sin atenuación por la cell state, y los gates aprenden qué olvidar, recordar y emitir.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar los 3 gates de LSTM: forget, input, output, y la cell state que viaja "horizontal".
- Diferenciar LSTM (3 gates + 2 estados) de GRU (2 gates + 1 estado, más simple, casi igual de bueno).
- Usar keras.layers.LSTM y keras.layers.GRU con return_sequences, return_state, recurrent_dropout.
- Aplicar Bidirectional cuando la tarea lo permite.

- Reconocer que con T > 100, incluso LSTM lucha — preferir Transformers.

Temas

- Cell state c_t (memoria larga) vs hidden state h_t (memoria corta).
- Forget gate: cuánto borrar de c_{t-1}.
- Input gate: cuánto agregar nuevo a c_t.
- Output gate: cuánto exponer en h_t.
- GRU: combina forget e input en una sola "update gate".
- Bidirectional + stacked LSTMs como receta clásica pre-Transformers.

Definiciones y características

- LSTM cell: 3 gates con sigmoid (0-1) que controlan flujo, + tanh para candidatos.
- GRU cell: update + reset gate. Menos parámetros que LSTM.
- recurrent_dropout: dropout aplicado a las conexiones recurrentes (entre timesteps).
- CuDNN kernel: LSTM/GRU tienen implementación cuDNN ultra rápida si cumplís restricciones (default activation tanh, recurrent activation sigmoid, no recurrent_dropout).
- Stacked LSTM: apilar varios → mejor capacidad pero más caro.

Dataset / recursos

- IMDB para sentimiento.
- Serie temporal del ejercicio anterior.
- Librerías: tensorflow, keras.

Ejercicios

1. LSTM vs SimpleRNN: en una serie de 100 pasos con dependencia long-range, comparar accuracy.
2. GRU vs LSTM: misma tarea, comparar. GRU ~ 25 % menos params; accuracy casi igual.
3. Stacked: 2-3 capas LSTM apiladas con return_sequences=True en las primeras.
4. Bidirectional: para sentimiento IMDB, comparar LSTM vs Bidirectional(LSTM).
5. cuDNN check: medir velocidad LSTM vanilla vs con recurrent_dropout (desactiva cuDNN).

Homework verificable

Clasificación de sentimiento sobre IMDB con LSTM:

1. Tokenizar con TextVectorization(max_tokens=20_000, output_sequence_length=200).
2. Embedding(20_000, 128) → Bidirectional(LSTM(64)) → Dense(64) → Dense(1, sigmoid).
3. Entrenar 5 épocas.
4. Reportar accuracy y comparar contra una baseline Dense sin RNN.

Criterio de aceptación: accuracy ≥ 0.85; el modelo recurrente claramente supera al baseline Dense.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
LSTM lento en GPU	Tal vez recurrent_dropout > 0 o unroll=Tru
Padding/masking ignorado	mask_zero=True en Embedding + LSTM que res
Bidirectional rompe en task causal (foreca	Bidirectional ve el futuro. Fix: solo Forw
Stacked LSTM overfittea	Demasiado expresivo. Fix: dropout entre ca
GRU vs LSTM comparados sin suficientes see	Diferencia chica puede ser ruido. Fix: ≥ 3

Preguntas frecuentes

¿LSTM o GRU?

Empezá con LSTM (más conocido). GRU si necesitás eficiencia. La diferencia de accuracy es típicamente < 0.5 pp.

¿LSTM aún se usa en 2026?

Sí, para forecasting de series temporales chicas, sistemas embedded, NER simple. Para NLP serio → Transformers.

¿Cuántas capas?

1-3. Más de 3 raramente justifica el costo. Para tareas serias, mejor un Transformer.

¿recurrent_dropout vs dropout?

dropout se aplica a inputs (entre capas); recurrent_dropout entre timesteps (intra-secuencia). El segundo desactiva cuDNN.

¿Cómo manejo secuencias de longitud variable?

Embedding(..., mask_zero=True) + padding con 0s. LSTM/GRU respetan la máscara automáticamente.

Referencias

- Géron, cap. 15 — Tackling Short-Term Memory Problems.
- Hochreiter & Schmidhuber (1997), Long Short-Term Memory, Neural Computation.
- Cho et al. (2014), Learning Phrase Representations using RNN Encoder–Decoder (GRU), EMNLP.
- Chung et al. (2014), Empirical Evaluation of Gated Recurrent Neural Networks.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 138 — Clase 138 — 1D CNNs y WaveNet

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 15 § Handling Long Sequences + WaveNet paper (van den Oord et al. 2016). Duración estimada: 60 min.*

Objetivo

Conocer la alternativa a RNN para secuencias: Conv1D y WaveNet (dilated causal convolutions). Más rápido que LSTM (paralelizable), receptive field amplio con pocas capas (dilated convolutions). Útil para audio, series temporales y como capa de preprocesamiento.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar Conv1D(filters, kernel_size, padding='causal') sobre secuencias.
- Implementar dilated convolutions (kernel salta posiciones).
- Reconocer causal convolution: el output en t depende solo de inputs ≤ t.
- Construir un mini-WaveNet con dilation rates exponenciales (1, 2, 4, 8, ...).

- Comparar Conv1D vs LSTM en speed/accuracy.

Temas

- Conv1D fundamentos: stride, padding, dilation.
- Causal padding: para no ver el futuro.
- Dilated convolutions: receptive field grande sin más layers.
- WaveNet: stack de dilated causal convolutions (1, 2, 4, ..., 512).
- Conv1D vs LSTM: paralelización, receptive field, parameter count.

Definiciones y características

- Conv1D: convolución sobre una sola dimensión espacial/temporal.
- padding='causal': padding asimétrico de modo que output[t] depende solo de input[0..t].
- dilation_rate: gap entre los elementos del kernel. dilation=2 con kernel=3 → ve t, t-2, t-4.
- Receptive field: rango temporal que afecta el output. Con dilated stack 1,2,4,8,16: 2^N + 1.
- WaveNet: red de Conv1D dilated causal stackeadas, originalmente para generación de audio.

Dataset / recursos

- Serie temporal del ejercicio 119.
- Audio simple (sintético: superposición de senos).
- Librerías: tensorflow, keras.

Ejercicios

1. Conv1D vs LSTM: forecasting de serie. Conv1D(32, 5, padding='causal') → Conv1D(32, 5, padding='causal') → Flatten → Dense(1). Comparar con LSTM equivalente.
2. Dilation rates: stack de 4 Conv1D con dilation_rate {1, 2, 4, 8}. Calcular receptive field.
3. Speed test: medir tiempo de training Conv1D vs LSTM para misma data. Conv1D suele ser 5-20× más rápido en GPU.
4. WaveNet mini: implementar stack de 10 Conv1D causal con dilations {1, 2, 4, ..., 512} para una serie larga.
5. Visualización del receptive field: para un output [t], marcar qué inputs lo afectan.

Homework verificable

Forecasting del dataset eléctrico (clase 119) con mini-WaveNet:

1. Stack de 6 Conv1D causal dilated con rates 1, 2, 4, 8, 16, 32.
2. Cada capa con 32 filtros, kernel 2.
3. Comparar MAE en test vs LSTM del ejercicio 119.

Criterio de aceptación: el WaveNet debe ser competitivo o mejor que LSTM, y notablemente más rápido en GPU.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
padding='same' en forecasting causal	Ve el futuro. Fix: padding='causal'.
Stack sin dilation → receptive field chico	Necesitás muchas capas. Fix: dilations exp
Conv1D + Flatten → Dense con muchos params	Modelo grande. Fix: GlobalAvgPool1D antes
Mezclar Conv1D no causal y causal	Inconsistente. Fix: si forecasting, todo c
Audio en [-1, 1] con Conv1D sin normalizar	Funcionar funciona, pero converge mejor co

Preguntas frecuentes

¿Conv1D supera a Transformer en algún caso?

Sí, en audio raw y series largas (>1000 pasos) con patrones locales. Transformer atención O(N²) en memoria.

¿WaveNet sigue siendo state-of-the-art?

En generación de audio, fue reemplazada por modelos de difusión (clase 133) y autoregresivos sobre tokens de audio (EnCodec + Llama-style).

¿Combinar Conv1D + LSTM/Transformer?

Sí. Patrón "convs como tokenizer" + Transformer arriba es estándar en audio (Wav2Vec, Whisper).

¿dilation_rate y kernel_size cómo se combinan?

receptive_field = (kernel_size - 1) * dilation_rate + 1 para una sola capa. Stack: suma de eso por capa.

¿Por qué Conv1D paralelizable?

Todas las posiciones se computan en paralelo (no hay dependencia temporal explícita como RNN). Ideal para GPU.

Referencias

- Géron, cap. 15 — Using 1D Convolutional Layers + WaveNet.
- van den Oord et al. (2016), WaveNet.
- Bai et al. (2018), An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 139 — Clase 139 — Generación de texto char-RNN

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 16 § Generating Shakespearean Text Using a Character RNN. Duración estimada: 70 min.*

Objetivo

Construir un modelo de lenguaje autoregresivo a nivel carácter — el ejercicio canónico de Karpathy 2015 sobre Shakespeare. Entender next-token prediction como tarea de pre-training (la base de todo LLM moderno), sampling con temperatura, y por qué char-RNN fue importante históricamente aunque hoy se hace con tokens BPE y Transformers.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Construir un vocabulario de caracteres (stoi, itos dicts).
- Generar samples (window, target) donde el target es el siguiente carácter.

- Entrenar un modelo Embedding → LSTM → Dense(vocab_size) con cross-entropy.
- Implementar sampling autoregresivo: softmax → multinomial → next char → feed back.
- Aplicar temperatura: logits / T antes de softmax — bajo T = más determinista, alto T = más random.

Temas

- Tokenización a nivel carácter vs word vs BPE.
- Stateful vs stateless RNN: stateful=True para mantener h entre batches.
- Loss: cross-entropy sobre vocab_size.
- Sampling: greedy vs categorical multinomial vs top-k vs nucleus.
- Temperatura como control de creatividad.

Definiciones y características

- Next-token prediction: predecir x_{t+1} dado x_1, ..., x_t. Self-supervised.
- Vocabulario: lista única de caracteres del corpus.
- Stateful RNN: hidden state persiste entre batches consecutivos. Útil para corpus largo.
- Sampling temperatura: p_i = softmax(logits / T). T < 1 más confiado; T > 1 más diverso.
- Top-k sampling: solo considerar los k tokens más probables. Default moderno con k=40-50.

Dataset / recursos

- Tiny Shakespeare (~1 MB de obras): https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt
- Librerías: tensorflow, keras, numpy.

Ejercicios

1. Vocab + encoding: tokenizar el texto a ints. Reportar vocab_size.
2. Modelo: Embedding(vocab_size, 64) → GRU(128, return_sequences=True) → Dense(vocab_size).
3. Train: pasar batches de longitud 100; loss = sparse_categorical_crossentropy.
4. Sample: implementar función que genera N caracteres con T=1.0, T=0.5, T=1.5. Comparar outputs.
5. Top-k: implementar np.argpartition(logits, -k) antes de muestrear.

Homework verificable

Generar Shakespeare:

1. Entrenar Embedding(64) → GRU(256, return_sequences=True, stateful=True) → Dense(vocab_size) por 10 épocas.
2. Generar 1000 caracteres comenzando con "ROMEO:" a temperaturas 0.3, 0.7, 1.2.
3. Reportar val_loss y muestras de output.

Criterio de aceptación: val_loss < 1.5 (vs ~3.5 random); las muestras a T=0.7 tienen palabras reconocibles y estructura de obra teatral.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Output es gibberish total	Modelo no entrena (loss alta) o T demasiad
Output es siempre la misma frase	T = 0 o muy bajo. Fix: T = 0.5 - 1.0.
OOM al generar 10 000 chars	Stateful + sin truncar window. Fix: usar w
argmax greedy → loops ("the the the the")	Greedy sin diversidad. Fix: sampling con t
Vocabulario incluye chars raros que aparec	Aumenta vocab innecesariamente. Fix: filtr

Preguntas frecuentes

¿char-RNN vs word-RNN vs BPE?

Char es simple, no tiene problema de OOV pero los outputs son largos. Word es eficiente pero pierde con palabras nuevas. BPE / SentencePiece (clase 127) es el estándar moderno.

¿Esto es un "LLM"?

Conceptualmente sí: next-token prediction. La diferencia con GPT es escala (parámetros, datos) y arquitectura (Transformer vs RNN). Mismo objetivo.

¿Sampling estrategias avanzadas?

Top-k, nucleus / top-p (Holtzman et al. 2020), beam search. Default moderno: top-p con p=0.9 + temperatura.

¿Cómo evaluar generación?

Perplexity (exp(loss)) automática. Para calidad subjetiva, evaluadores humanos o métricas como BLEU, ROUGE, BERTScore.

¿Char-RNN sirve para algo en 2026?

Sí: NER muy específico, dominios con vocabulario muy chico, o como baseline. Para generación de texto serio → Transformer + BPE.

Referencias

- Géron, cap. 16 — Generating Shakespearean Text.
- Karpathy (2015), The Unreasonable Effectiveness of Recurrent Neural Networks (blog).
- Holtzman et al. (2020), The Curious Case of Neural Text Degeneration (nucleus sampling), ICLR.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 140 — Clase 140 — Análisis de sentimiento

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 16 § Sentiment Analysis. Duración estimada: 65 min.*

Objetivo

Aplicar un modelo de clasificación de texto sobre IMDB reviews — la tarea NLP más clásica para benchmarks. Pipeline completo: TextVectorization → Embedding → arquitectura (Dense / CNN / RNN / Transformer) → Dense(1, sigmoid). Comparar el zoo de approaches y reconocer que con Hugging Face hoy se hace en 3 líneas (clase 127).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Tokenizar y vectorizar texto con TextVectorization(max_tokens=20_000, output_sequence_length=200).
- Aplicar Embedding(vocab_size, dim) y entender que es una lookup table aprendible.
- Construir 4 arquitecturas: bag-of-embeddings (sin orden), Conv1D, LSTM, Bidirectional LSTM.

- Comparar accuracy de las 4 vs un baseline TfidfVectorizer + LogisticRegression.
- Usar Embedding(..., mask_zero=True) para manejar padding correctamente.

Temas

- TextVectorization moderno (Keras 3+).
- Embedding: lookup table inicializada random y entrenable.
- BagOfEmbeddings (mean pooling) como baseline DL.
- Conv1D para texto: capta n-gramas.
- LSTM + Bidirectional: captura contexto largo y bidireccional.
- Pre-trained embeddings (GloVe, Word2Vec) — históricamente importantes; hoy reemplazados por embeddings de transformers.

Definiciones y características

- Embedding: matriz (vocab_size, embed_dim). La fila i es la representación aprendida del token i.
- Pooling: tras Embedding tenés (batch, T, dim); pooling reduce a (batch, dim). Mean, max, attention.
- Masking: ignorar posiciones con padding (0 por convención).
- Bidirectional: corre LSTM forward + backward y concatena.

Dataset / recursos

- keras.datasets.imdb.load_data() o tfds.load('imdb_reviews').
- Librerías: tensorflow, keras.

Ejercicios

1. Baseline ML clásico: TfidfVectorizer + LogisticRegression. Accuracy de referencia (~0.88).
2. Bag-of-embeddings: Embedding → GlobalAveragePooling1D → Dense(1, sigmoid). Reportar accuracy.
3. Conv1D: Embedding → Conv1D(64, 5) → GlobalMaxPool1D → Dense(1, sigmoid).
4. Bidirectional LSTM: Embedding → Bidirectional(LSTM(64)) → Dense(1, sigmoid).
5. Pre-trained: cargar GloVe 100d y inicializar la matriz de Embedding con ellos. Comparar accuracy contra inicialización random.

Homework verificable

IMDB sentiment classifier con 3 arquitecturas:

1. Pipeline TextVectorization(20_000, 200).
2. Tres modelos: Conv1D, Bidirectional LSTM, BagOfEmbeddings.
3. Reportar accuracy en test para cada uno.
4. Comparar contra TFIDF + LogReg.

Criterio de aceptación: al menos uno de los modelos DL debe ≥ 0.88. Bidirectional LSTM suele ganar pero por margen pequeño vs Conv1D.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
output_sequence_length muy chico	Trunca reviews largos. Fix: 200-500 para r
Olvido mask_zero=True en Embedding	LSTM/GRU aún ven el padding como token vál
Sin pooling antes del Dense → shapes incom	Fix: GlobalAveragePooling1D().
GloVe sin escalado de dimensiones	Embeddings vienen pre-trained con norma es
Reviews en otros idiomas con tokenizer ent	Mal performance. Fix: usar un tokenizer mu

Preguntas frecuentes

¿Sentimiento en 2026: TextVectorization o HF?

Para producción serio: HuggingFace + distilbert-base-uncased-finetuned-sst-2-english (94 % accuracy, 5 líneas de código). Esta clase es pedagógica.

¿Embeddings pre-trained todavía importan?

Para tokens "raw" no — todo modelo moderno tiene su propio embedding integrado. Para visualizar relaciones semánticas (analogías), sí.

¿Cuántos tokens y cuánta longitud?

IMDB: max_tokens=20_000 cubre vocab; output_sequence_length=200 cubre el 90 % de reviews.

¿Bidirectional siempre mejor?

Para clasificación (non-causal), sí. Para generación o real-time, NO (necesitarías ver el futuro).

¿Sentimiento multi-clase (estrellas 1-5)?

Cambiar última capa a Dense(5, softmax) + sparse_categorical_crossentropy. Para sentiment ordinal, considerar ordinal regression.

Referencias

- Géron, cap. 16 — Sentiment Analysis.
- Maas et al. (2011), Learning Word Vectors for Sentiment Analysis — IMDB dataset.
- Pennington et al. (2014), GloVe, EMNLP.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 141 — Clase 141 — Encoder-Decoder para traducción

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 16 § Encoder–Decoder Network for Neural Machine Translation. Duración estimada: 80 min.*

Objetivo

Implementar la arquitectura seq2seq (Sutskever, Vinyals & Le 2014) — un encoder que comprime la oración fuente en un vector de contexto + un decoder que genera la oración destino token por token. Conocer teacher forcing (durante training, el decoder ve los targets reales como input) vs inference autoregresiva. Esta arquitectura es la antesala de atención (clase 125) y de Transformers (clase 126).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Construir un seq2seq con dos LSTM: encoder devuelve state, decoder usa ese state como inicialización.
- Aplicar teacher forcing: en training, decoder input = target shifted; en inference, autoregresivo.

- Tokens especiales: <start>, <end>, <pad>, <unk>.
- Reconocer la limitación del cuello de botella (todo el meaning de la oración fuente en un vector fijo) — motivación para atención.
- Evaluar traducción con BLEU (nltk.translate.bleu_score).

Temas

- Arquitectura clásica seq2seq con 2 LSTM.
- Teacher forcing vs scheduled sampling.
- Inference autoregresiva con generate loop.
- Beam search (mejor que greedy).
- BLEU como métrica.
- Limitación del bottleneck → motivó atención (Bahdanau 2014).

Definiciones y características

- Encoder: LSTM que procesa la fuente, devuelve final_state = (h_T, c_T).
- Decoder: LSTM inicializado con final_state del encoder; genera target.
- Teacher forcing: durante training, decoder recibe el target real (shifted) en lugar de su propia predicción.
- <start> / <end> tokens: marcadores para iniciar/parar generación.
- BLEU: métrica de calidad de traducción basada en n-grams overlapping. Va de 0 a 100.
- Beam search: durante inference, mantiene los k mejores prefijos en lugar de greedy.

Dataset / recursos

- Tatoeba English-Spanish (small): https://www.manythings.org/anki/
- Librerías: tensorflow, keras, nltk (para BLEU).

Ejercicios

1. Preparar datos: tokenizar source y target, agregar <start> y <end> al target, padding.
2. Encoder: Embedding → LSTM(256, return_state=True). Mantener state_h, state_c.
3. Decoder en training: Embedding(decoder_input) → LSTM(256, initial_state=encoder_state) → Dense(target_vocab, softmax).
4. Inference loop: feed <start>, predecir, alimentar prediction como next input, hasta <end> o max_len.
5. BLEU: calcular sobre el test set.

Homework verificable

Traducción inglés → español con seq2seq:

1. Dataset Tatoeba (~10k frases).
2. Encoder + decoder LSTM con 256 units.
3. Train 30 épocas con teacher forcing.
4. Inference autoregresiva + BLEU.

Criterio de aceptación: BLEU > 10 en test (modesto pero válido para LSTM básico); muestras de traducción reconocibles aunque imperfectas.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Modelo genera <end> inmediatamente	El bias del decoder lleva siempre a <end>.
Greedy inference repite palabras	"the the the". Fix: beam search o repetiti
Inference más lento que esperado	Loop Python en cada token. Fix: usar tf.fu

BLEU = 0	Mismatch en tokens entre prediction y refe
Vocabulario target muy grande	Sample softmax o tied embeddings. Fix: lim

Preguntas frecuentes

¿seq2seq aún se usa?

Conceptualmente sí (Transformer es seq2seq con atención). Como RNN-seq2seq pure: solo histórico o casos muy específicos. Para traducción seria → Transformer + Hugging Face (clase 127).

¿Bottleneck del estado fijo?

Una sola tupla (h, c) para resumir 50 tokens de input es muy pobre. Atención (clase 125) resuelve dando al decoder acceso a TODOS los estados del encoder.

¿Teacher forcing vs scheduled sampling?

Teacher forcing = siempre target real. Funciona, pero hay mismatch entre train y inference. Scheduled sampling alterna entre target y predicción propia → mejor.

¿Cómo manejo vocabulario grande?

BPE / SentencePiece (clase 127). Vocab típico 32k subwords cubre cualquier idioma con OOV manejable.

¿BLEU es buena métrica?

Aceptable para traducción "literal". Falla con paráfrasis. Para evaluación humana o LLM-as-judge, mejores opciones.

Referencias

- Géron, cap. 16 — Encoder-Decoder Network for Neural Machine Translation.
- Sutskever, Vinyals & Le (2014), Sequence to Sequence Learning with Neural Networks, NeurIPS.
- Cho et al. (2014), Learning Phrase Representations using RNN Encoder-Decoder.
- Papineni et al. (2002), BLEU, ACL.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 142 — Clase 142 — Mecanismos de atención

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 16 § Attention Mechanisms + Bahdanau et al. (2015), Luong et al. (2015). Duración estimada: 75 min.*

Objetivo

Entender la atención — el mecanismo que destrabó NLP moderno. Bahdanau (2015): permite al decoder mirar todos los hidden states del encoder, ponderando dinámicamente. Luego self-attention (Vaswani 2017): tokens dentro de la misma secuencia se atienden entre sí → Transformer (clase 126).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar scaled dot-product attention: softmax(QK^T / √d) V.
- Diferenciar cross-attention (decoder→encoder, clásico Bahdanau) de self-attention (intra-secuencia, base del Transformer).
- Implementar attention a mano en numpy/TF para una secuencia corta.
- Aplicar keras.layers.Attention o keras.layers.MultiHeadAttention.
- Interpretar attention weights como "qué tokens fuente miró el decoder al generar cada token destino".

Temas

- Motivación: bottleneck del encoder en seq2seq.
- Bahdanau (additive) vs Luong (multiplicative / dot-product).
- Q, K, V: queries, keys, values.
- Scaled dot-product attention.
- Self-attention vs cross-attention.
- Multi-head: paralelizar varias attention heads con subspaces distintos.
- Attention weights visualizables.

Definiciones y características

- Query (Q): "lo que estoy buscando" (e.g., el token del decoder).
- Key (K): "lo que cada elemento ofrece como índice" (e.g., los hidden del encoder).
- Value (V): "el contenido a recuperar" (normalmente igual o relacionado a K).
- Attention weights: softmax(QK^T / √d), matrix (seq_len_q, seq_len_k).
- Scaled: dividir por √d para que el softmax no sature.
- Multi-head: dividir d en h heads, calcular attention por separado, concatenar.

Dataset / recursos

- Reusar dataset de traducción de 124.
- Librerías: tensorflow, keras.

Ejercicios

1. Attention a mano: Q, K, V random (seq, d); calcular softmax(QK^T / √d) V. Verificar shapes.
2. Visualizar attention map: tras entrenar un seq2seq con atención, plot heatmap de los pesos (target, source) para una traducción.
3. MultiHeadAttention: mha = MultiHeadAttention(num_heads=8, key_dim=64); output = mha(query, value).
4. Self-attention: aplicar mha(x, x) (query = key = value). Esto es el bloque de Transformer.
5. Causal mask: para generación autoregresiva, MultiHeadAttention(..., use_causal_mask=True).

Homework verificable

Traducción con atención sobre el dataset de 124:

1. Encoder LSTM con return_sequences=True.
2. Decoder con cross-attention sobre todos los outputs del encoder.
3. Train y evaluar BLEU.
4. Visualizar attention map para 2 frases.

Criterio de aceptación: BLEU debe mejorar respecto al seq2seq sin atención (de 124) por ≥ 5 pp; el attention map muestra alineamiento source-target razonable.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Olvido scale / √d	Softmax satura con d grande. Fix: usar Mul
Attention sin masking sobre padding	Atiende a tokens <pad>. Fix: attention_mas
Causal mask al revés	Mira el futuro. Fix: use_causal_mask=True
Cross-attention con Q de un tamaño y K/V d	OK si d_q != d_kv, pero MultiHeadAttention
Visualizar attention de un modelo no conve	Mapas ruidosos. Fix: entrenar bien primero

Preguntas frecuentes

¿Self vs cross attention?

Self: Q = K = V (la secuencia se atiende a sí misma). Cross: Q de un origen, K/V de otro (decoder mira encoder).

¿Por qué √d y no d en el scaled?

QK^T tiene varianza ~ d. Dividir por √d mantiene varianza estable y softmax no satura.

¿Multi-head qué aporta?

Cada head puede aprender un patrón distinto (one por sintaxis, otro por semántica, ...). Empíricamente importante.

¿Attention O(N²)?

Sí, en memoria y compute. Es el principal limitante para secuencias largas. Flash Attention (clase 126) lo optimiza; Linformer/Performer aproximan.

¿Atención biológica?

Inspiración loose. El mecanismo no es realmente cómo funciona el cerebro, pero la metáfora "prestar atención a partes relevantes" es intuitiva.

Referencias

- Géron, cap. 16 — Attention Mechanisms.
- Bahdanau, Cho & Bengio (2015), Neural Machine Translation by Jointly Learning to Align and Translate, ICLR.
- Luong, Pham & Manning (2015), Effective Approaches to Attention-based Neural Machine Translation, EMNLP.
- Vaswani et al. (2017), Attention Is All You Need, NeurIPS — paper del Transformer.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 143 — Clase 143 — Transformers: arquitectura, BERT, GPT (+ Flash Attention, RoPE, GQA)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 16 § The Transformer Architecture + Vaswani et al. (2017) + papers BERT, GPT, FlashAttention. Duración estimada: 100 min.*

Objetivo

Dominar la arquitectura Transformer —encoder, decoder, ambas variantes (BERT encoder-only, GPT decoder-only, T5 encoder-decoder)— a nivel de poder implementarla a mano. Conocer las mejoras clave 2022-2024 que hacen a los LLMs modernos rápidos y eficientes: Flash Attention v2/v3, RoPE (Rotary Position Embeddings), Grouped-Query Attention (GQA).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Dibujar el block de Transformer: LayerNorm → MultiHeadAttention → +residual → LayerNorm → FFN → +residual.
- Implementar positional encoding sinusoidal o aprendible.
- Reconocer 3 variantes: encoder-only (BERT), decoder-only (GPT), encoder-decoder (T5, Whisper).
- Saber qué hace cada mejora moderna: Flash Attention (O(N) memoria, 2-3× speedup), RoPE (mejor extrapolación a secuencias largas), GQA (compartir KV heads → menos KV cache en inference).
- Cargar y usar un Transformer chico con keras.layers.MultiHeadAttention o desde Hugging Face.

Temas

- Block Transformer: LN → MHA → res → LN → FFN → res.
- Positional encoding: por qué se necesita (attention es permutation-invariant) — sin → aprendible → RoPE.
- BERT: encoder-only, MLM + NSP, bidireccional.
- GPT: decoder-only, next-token, causal mask.
- T5 / BART: encoder-decoder, span-corruption / denoising.
- Complemento moderno: Flash Attention, RoPE, GQA, MQA.

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 126b — Flash Attention v2/v3, RoPE, GQA: el motor de los LLMs modernos

Definiciones y características

- Transformer block: bloque básico repetido N veces.
- MHA (Multi-Head Attention): H heads paralelos, cada uno con Q, K, V proyectados.
- Positional encoding: información de orden inyectada en embeddings.
- BERT: encoder-only, bidirectional, masked LM pre-training.
- GPT: decoder-only, causal mask, autoregressive pre-training.
- T5: encoder-decoder, span-corruption pre-training.
- Flash Attention: implementación memory-efficient de scaled-dot-product attention.
- RoPE: rotación de Q/K en función de posición.
- GQA: heads de KV compartidos entre grupos de heads de Q.
- KV cache: almacenamiento de K y V de tokens previos para inference autoregresiva eficiente.

Dataset / recursos

- WikiText-2 o similar.
- Modelos preentrenados desde HF (clase 127).
- Librerías: tensorflow, keras, transformers.

Ejercicios

1. Transformer block desde cero: implementar def transformer_block(x): x = x + mha(LN(x)); x = x + ffn(LN(x)); return x con MultiHeadAttention y FFN.
2. Positional encoding sinusoidal: implementar la fórmula original de Vaswani; visualizar como heatmap.
3. Mini-GPT: 4 capas Transformer con causal mask. Entrenar en next-token sobre Tiny Shakespeare. Comparar con char-RNN (122).
4. RoPE manual: implementar la rotación. Aplicar y verificar que attention(Q_rot, K_rot) da diferencia relativa.
5. HuggingFace check: cargar bert-base-uncased y gpt2; imprimir model.config. Identificar num_attention_heads, hidden_size, intermediate_size.

Homework verificable

Entrenar un mini-GPT desde cero sobre Tiny Shakespeare:

1. 4 capas Transformer decoder con causal mask.
2. d_model=128, n_heads=4, vocab=char-level.
3. Train 20 épocas; generar 1000 caracteres con temperatura 0.7.
4. Comparar val_loss y muestras vs char-RNN (clase 122).

Criterio de aceptación: val_loss < 1.3 (mejor que char-RNN); samples más coherentes con estructura de obra teatral.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Modelo entrena pero outputs son random	Olvido del causal mask en decoder. Fix: us
LayerNorm antes vs después: dudas	"Pre-LN" (antes) entrena mucho mejor que "
Positional encoding aprendible explota	Init mal. Fix: small std (0.02) o usar RoP
Sin gradient checkpointing → OOM en secuen	Fix: tf.recompute_grad o gradient checkpoi
FFN sin expansión	Default es d_ff = 4 * d_model. Fix: respet

Preguntas frecuentes

¿Encoder-only o decoder-only o ambos?

Encoder-only (BERT): clasificación, NER, similaridad. Decoder-only (GPT): generación, LLMs modernos. Encoder-decoder (T5, Whisper): traducción, transcripción.

¿GPT-style "es todo lo que necesitás" hoy?

Para LLMs generales sí. BERT-style sigue siendo eficiente para clasificación.

¿LayerNorm o RMSNorm?

RMSNorm (Zhang & Sennrich 2019) es estándar moderno — más rápida sin pérdida de calidad. Lo usan Llama, Mistral, etc.

¿Por qué RoPE en lugar de aprendible?

(a) Mejor extrapolación a longitudes no vistas, (b) propiedad relativa útil, (c) no agrega parámetros.

¿GQA solo para inference?

Se entrena con GQA desde el principio. Beneficio principal es inference; training también es marginalmente

más rápido.

Referencias

- Géron, cap. 16 — The Transformer Architecture.
- Vaswani et al. (2017), Attention Is All You Need, NeurIPS.
- Devlin et al. (2018), BERT, NAACL.
- Radford et al. (2018-2019), GPT, GPT-2, OpenAI.
- Dao et al. (2022, 2023, 2024), FlashAttention v1/v2/v3.
- Su et al. (2021), RoFormer: Enhanced Transformer with Rotary Position Embedding.
- Ainslie et al. (2023), GQA: Training Generalized Multi-Query Transformer Models.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 144 — Clase 144 — Flash Attention v2/v3, RoPE, GQA: el motor de los LLMs modernos

> *Parte: 2 — Deep Learning · Fuente: Dao et al. (2022, 2023, 2024) FlashAttention + Su et al. (2021) RoPE + Ainslie et al. (2023) GQA. Duración estimada: 90 min.*

Objetivo

Entender en profundidad las 3 piezas técnicas que hacen que un LLM moderno (Llama 3, Mistral, Qwen, Gemma) sea rápido y memory-efficient: Flash Attention v2/v3 (O(N) memoria + 2-3× speedup), Rotary Position Embeddings (RoPE) (mejor extrapolación), Grouped-Query Attention (GQA) (menos KV cache en inference).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar por qué attention naïve es O(N²) en memoria y cómo FlashAttention lo reduce a O(N) con online softmax + tiling.
- Implementar RoPE: rotar pares de dimensiones de Q, K por ángulo función de posición.
- Diferenciar MHA, MQA, GQA — y por qué GQA es el compromiso default 2026.
- Aplicar torch.nn.functional.scaled_dot_product_attention(q, k, v, is_causal=True) que elige Flash auto.
- Reconocer combinación moderna: RMSNorm + GQA + RoPE + SwiGLU + Flash Attention.

Temas

- Attention cost: matriz (N, N) → 64 MB por head con N=8192, fp16.
- FlashAttention: bloques en SRAM, no materializa la matriz completa.
- v1 (2022), v2 (2023, 2× speedup), v3 (2024, optimizado H100).
- Positional encoding: sinusoidal → learnable → RoPE.
- RoPE: rotación bidimensional, propiedad relativa.
- MHA / MQA / GQA: trade-off entre calidad y memoria.
- KV cache: por qué crece en inference.

Definiciones y características

- FlashAttention: algoritmo IO-aware. Reformula softmax(QK^T)V en bloques que caben en SRAM.
- Online softmax: actualización incremental del softmax sin materializar todo.
- RoPE: q' = R_θ q donde R_θ rota pares de dims. θ_i = 10000^(-2i/d).
- MHA: H_q = H_kv (clásico).
- GQA: H_kv = H_q / G. G grupos. Llama 2 70B usa G=8.
- MQA: H_kv = 1. Extremo de GQA.

Dataset / recursos

- HuggingFace modelos: Llama 3, Mistral 7B.
- Librerías: flash-attn, torch ≥ 2.0 (SDPA), transformers.

Ejercicios

1. SDPA vs naïve: implementar attention naïve y F.scaled_dot_product_attention. Benchmark.
2. RoPE: implementar rotation function, verificar propiedad attention(R_θ q, R_φ k) = f(θ - φ).
3. GQA Vs MHA: con Llama config (n_heads=32, kv_heads=8), inspeccionar shapes.
4. KV cache: medir VRAM en inference con secuencia 8192 — comparar MHA vs GQA.
5. FlashAttention v3 en H100: si tenés H100, benchmark vs v2.

Homework verificable

Mini-GPT con piezas modernas:

1. 6-layer Transformer con: RMSNorm, GQA (4 KV heads / 8 Q heads), RoPE, SwiGLU FFN.
2. Train next-token sobre Tiny Shakespeare.
3. Comparar contra mini-GPT clásico (LayerNorm + MHA + Sin PE + GELU FFN).

Criterio de aceptación: mini-GPT moderno entrena más estable + menor memoria; quality comparable.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
flash-attn no instala en CUDA viejo	Requiere CUDA 11.6+, GPU Ampere+. Fix: usa
RoPE con base distinta a 10000	Para extrapolación a contextos largos (32k
MQA da peor calidad	Esperado. Fix: GQA es el compromiso.
KV cache OOM en context largo	Inherente. Fix: GQA + quantization (Q8 KV)
is_causal=True en SDPA solo aplica si tens	Fix: passing attn_mask cuando shapes asimé

Preguntas frecuentes

FlashAttention v2 o v3?

v3 si tenés H100. v2 estable para todo lo demás. SDPA de PyTorch elige el mejor disponible.

RoPE absoluto o relativo?

RoPE codifica posición absoluta pero produce comportamiento relativo en attention. Lo mejor de ambos.

GQA en training?

Sí — entrenar con GQA desde el principio. Llama 2 70B y todos los modernos lo hacen.

Combina con sliding window attention?

Sí — Mistral 7B usa GQA + sliding window. Para contextos infinitos.

¿Y para CV (ViT)?

ViT moderno también usa Flash Attention (timm support). RoPE en algunos (DiT). GQA menos común en CV.

Referencias

- Dao et al. (2022, 2023, 2024), FlashAttention v1/v2/v3.
- Su et al. (2021), RoFormer: Enhanced Transformer with Rotary Position Embedding.
- Ainslie et al. (2023), GQA: Training Generalized Multi-Query Transformer Models.
- Touvron et al. (2023), Llama 2.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 145 — Clase 145 — Hugging Face Transformers (uso práctico)

> *Parte: 2 — Deep Learning · Fuente: HuggingFace docs + Géron, cap. 16 § Pretrained Transformer Models. Duración estimada: 80 min.*

Objetivo

Dominar Hugging Face Transformers — la librería estándar de la industria para usar modelos preentrenados (BERT, GPT, T5, Llama, Whisper, ViT...). Aprender el pipeline API (one-liner para 90 % de los casos), el Trainer API (fine-tuning con muy poco código) y los componentes manuales (Tokenizer + Model + DataLoader).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Usar pipeline('sentiment-analysis'), pipeline('summarization'), pipeline('zero-shot-classification'), etc. en 1 línea.
- Cargar manualmente: tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased'), model = AutoModelForSequenceClassification.from_pretrained('...').
- Tokenizar con tokenizer(texts, padding=True, truncation=True, return_tensors='pt').
- Fine-tunear con Trainer sobre un dataset propio.
- Reconocer el Hub (huggingface.co/models) como catálogo de + de 500k modelos.

Temas

- pipeline API: lo más fácil. Tareas: sentiment, NER, QA, summarization, translation, fill-mask, zero-shot, etc.
- AutoTokenizer + AutoModel*: API manual flexible.
- Tokenizers: BPE, WordPiece, SentencePiece, tiktoken.
- Trainer + TrainingArguments: fine-tuning con 20 líneas.
- Hub: descubrir modelos, datasets, spaces.
- datasets library: cargar datasets, preprocesar.

Definiciones y características

- AutoTokenizer: detecta el tokenizer adecuado para el modelo.
- AutoModelFor<Task>: cabezas específicas (SequenceClassification, TokenClassification, CausalLM, Seq2SeqLM).
- Subword tokenization (BPE/WP/SP): tokens son piezas de palabras → vocab fijo (~32-64k), nada de OOV.
- Special tokens: [CLS], [SEP], <s>, </s>, [PAD], <|im_start|>, etc. Dependen del modelo.
- Hub: GitHub para modelos — cualquiera puede subir/descargar.

Dataset / recursos

- HuggingFace Hub: <https://huggingface.co/models>.
- Dataset: load_dataset('imdb'), load_dataset('squad'), etc.
- Librerías: transformers, datasets, accelerate, evaluate.

Ejercicios

1. pipeline one-liner: pipe = pipeline('sentiment-analysis'); pipe('I loved this movie!'). Inspeccionar output.
2. Zero-shot classification: pipeline('zero-shot-classification')(text, candidate_labels=['sports', 'politics', 'tech']). Sin training específico.
3. Manual: tokenizar texto con bert-base-uncased. Inspeccionar input_ids, attention_mask, token_type_ids.
4. Modelo + forward: outputs = model(**inputs). Inspeccionar outputs.logits.
5. Tokenizer especiales: tokenizer.decode([101, 7592, 102]) → '[CLS] hello [SEP]'.

Homework verificable

Fine-tuning de distilbert-base-uncased sobre IMDB:

1. load_dataset('imdb').
2. Tokenizar con tokenizer(texts, truncation=True, padding='max_length', max_length=256).
3. AutoModelForSequenceClassification.from_pretrained(..., num_labels=2).
4. TrainingArguments(output_dir, num_train_epochs=2, per_device_train_batch_size=16, eval_strategy='epoch').
5. Trainer(...).train().
6. Reportar accuracy en test.

Criterio de aceptación: accuracy ≥ 0.92 (DistilBERT con 2 épocas alcanza ~0.93 en IMDB).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
OSError: Can't load tokenizer	Modelo no existe o sin internet. Fix: veri
RuntimeError: CUDA out of memory	Modelo + batch demasiado grandes. Fix: red
Tokenizer no agrega special tokens automát	Pasar add_special_tokens=True (default). S
Trainer no usa GPU	Default lo detecta; sino TrainingArguments
Fine-tuning de un modelo grande catastroph	LR alto. Fix: learning_rate=2e-5 o menor p

Preguntas frecuentes

¿pipeline o Trainer o AutoModel?

pipeline para usar un modelo preentrenado tal cual. AutoModel para custom training loop. Trainer para fine-tuning estándar (95 % de los casos).

¿PyTorch o TF backend?

PyTorch es estándar en HF — más modelos disponibles, mejor soportado. transformers soporta ambos pero los modelos modernos (Llama, Mistral) son solo PyTorch.

¿Tokenizer rápido?

AutoTokenizer.from_pretrained(..., use_fast=True) (default). Implementado en Rust, 10-100× más rápido que la versión Python.

¿Cómo descubro qué modelo usar?

<https://huggingface.co/models> con filtros por task, language, license. Modelos más descargados suelen ser buena apuesta.

¿Hugging Face Inference API en producción?

Sí, para prototipos. Para producción serio, self-host con transformers o vLLM / TGI (clase 139).

Referencias

- Hugging Face Transformers docs.
- Hugging Face course — el mejor recurso gratuito para empezar.
- Wolf et al. (2020), Transformers: State-of-the-Art Natural Language Processing, EMNLP demo.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 146 — Clase 146 — CLIP, SigLIP: multimodal embeddings (visión + texto)

> *Parte: 2 — Deep Learning · Fuente: Radford et al. (2021) CLIP + Zhai et al. (2023) SigLIP. Duración estimada: 80 min.*

Objetivo

Conocer CLIP (OpenAI 2021) y su evolución SigLIP (Google 2023) — los foundation models que mapean imágenes y texto al mismo espacio vectorial, entrenados con contrastive learning sobre 400M-4B pares. Aplicaciones: zero-shot classification, image search por texto, content moderation, embeddings para RAG multimodal.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Cargar CLIP/SigLIP desde HuggingFace: CLIPModel.from_pretrained('openai/clip-vit-base-patch32').
- Calcular embeddings de imágenes y de texto; cosine similarity entre ambos.
- Implementar zero-shot classification: predecir clase con la mayor similaridad a "a photo of a [class]".
- Hacer image retrieval por texto sobre un corpus de imágenes.
- Diferenciar CLIP (softmax contrastive) de SigLIP (sigmoid pairwise, mejor escalabilidad).

Temas

- Contrastive Language-Image Pre-training: matchear pares correctos, separar incorrectos.
- Arquitectura: image encoder (ViT) + text encoder (Transformer).
- Cosine similarity como métrica.
- Zero-shot vs few-shot.
- Variantes modernas: SigLIP (sigmoid loss), EVA-CLIP, OpenCLIP, Apple AIM.

Definiciones y características

- CLIP: dual encoder, contrastive softmax.
- SigLIP: dual encoder, sigmoid pairwise — escala mejor, no necesita global batch.
- Embedding space: cosine similarity para comparar.
- Zero-shot: clasificar sin training en la tarea, solo con prompt textual.
- Variantes: ViT-B/32 (rápido), ViT-L/14 (mejor), ViT-H/14 (top).

Dataset / recursos

- Imágenes propias o tfds.load('cats_vs_dogs').
- HuggingFace: openai/clip-vit-base-patch32, google/siglip-base-patch16-224.
- Librerías: transformers, torch, PIL.

Ejercicios

1. CLIP setup: cargar processor + model. Embed una imagen y un texto. Cosine similarity.
2. Zero-shot classification: dada una imagen, comparar contra ["a photo of a cat", "a photo of a dog"]. Predict argmax.
3. Image search: corpus de 100 imágenes; query texto "a sunset over the ocean" → top-5 más similares.
4. SigLIP: misma tarea, comparar accuracy.
5. Fine-tune ligero: con dataset chico custom, fine-tunear CLIP con LoRA para un dominio específico.

Homework verificable

Buscador de imágenes por texto:

1. 200 imágenes diversas.
2. Embed todas con CLIP ViT-B/32.
3. UI simple (notebook): input texto → top-5 imágenes.
4. Probar 10 queries; reportar precisión subjetiva.

Criterio de aceptación: queries claras devuelven imágenes relevantes en top-3 con frecuencia alta.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Similitud baja para imagen y texto que par	Prompt mal estructurado. Fix: "a photo of
Lento en CPU	Modelos grandes. Fix: ViT-B/32 (chico) o G
OOM con ViT-L/14	Fix: ViT-B/32 o batch=1.
Embeddings no normalizados → cosine raro	Fix: normalizar con outputs.image_embeds /
Imágenes en formato wrong	CLIP espera RGB 224×224. Fix: usar el proc

Preguntas frecuentes

CLIP o SigLIP?

SigLIP entrena más rápido y escala mejor; calidad similar. En 2026, SigLIP / SigLIP-2 es default moderno.

¿En LLM multimodal (LLaVA)?

CLIP/SigLIP es el image encoder. La LLM (Llama) recibe los embeddings.

Open vocabulary detection con CLIP?

OWL-ViT, Grounding DINO usan CLIP como base. Hacen detección con prompts texto.

CLIP en producción de búsqueda?

Sí — Pinterest, Adobe, Shopify. Vector DB (Qdrant, Pinecone) con embeddings CLIP.

Train CLIP desde cero?

400M+ pares necesarios. Open-source: OpenCLIP, MetaCLIP — alternativas abiertas con datasets públicos.

Referencias

- Radford et al. (2021), CLIP, ICML.
- Zhai et al. (2023), Sigmoid Loss for Language Image Pre-Training (SigLIP), ICCV.
- OpenCLIP.
- LAION-5B — dataset abierto.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 147 — Clase 147 — Whisper: ASR, transcripción, traducción de audio

> *Parte: 2 — Deep Learning · Fuente: Radford et al. (2022) Whisper + OpenAI release notes. Duración estimada: 70 min.*

Objetivo

Usar Whisper (OpenAI 2022, open-source) — el modelo de ASR (Automatic Speech Recognition) multilenguaje que destronó a Google STT y AWS Transcribe en accuracy. Cubrir transcripción, traducción a inglés, timestamps, word-level timing. Alternativas modernas: Whisper-large-v3, distil-whisper (4× más rápido), insanely-fast-whisper.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Cargar Whisper con transformers (openai/whisper-large-v3) o openai-whisper (lib oficial).
- Transcribir audio en cualquier idioma (99+ soportados).
- Aplicar task='translate' para traducir directo a inglés.
- Obtener timestamps a nivel palabra para subtítulos.
- Usar distil-whisper para inference 4-6× más rápida con calidad similar.

Temas

- Arquitectura: encoder-decoder Transformer + spectrogram input.
- Tamaños: tiny, base, small, medium, large-v3.

- Languages: detectado automático o explícito.
- Tareas: transcribe (en idioma origen), translate (→ inglés).
- Diarization (quién habla): no built-in, requiere pyannote.audio separado.
- Long-form audio: chunking con overlap.

Definiciones y características

- Whisper: encoder-decoder Transformer entrenado en 680k horas multi-lenguaje.
- pipeline('automatic-speech-recognition'): API HF más simple.
- Distil-Whisper: destillación 4× más rápida; calidad casi igual.
- WER (Word Error Rate): métrica estándar ASR.
- Timestamps: por segmento (default) o por word (con flag).

Dataset / recursos

- Cualquier audio: voz, podcasts, llamadas.
- HuggingFace: openai/whisper-large-v3, distil-whisper/distil-large-v3.
- Librerías: transformers, torch, librosa (procesamiento audio).

Ejercicios

1. Transcripción básica: cargar pipeline('asr', model='openai/whisper-base'); pasar un audio.
2. Multilenguaje: audio en español → transcribir; verificar.
3. Traducción: pipe(audio, task='translate') → texto en inglés.
4. Timestamps: pipe(audio, return_timestamps='word') → palabras con start/end seconds.
5. Distil-Whisper: comparar tiempo y WER vs full Whisper.

Homework verificable

Sistema de subtítulos:

1. Audio de 5-10 min (clase grabada, podcast).
2. Whisper-large-v3 con return_timestamps='word'.
3. Generar SRT (formato subtítulos) a partir del output.
4. Verificar manualmente la calidad en 1 minuto random.

Criterio de aceptación: SRT bien formado; timestamps razonables; ≥ 90 % de palabras correctas.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Audio en formato no soportado	Whisper espera 16kHz mono. Fix: librosa.lo
Lento en CPU	Whisper-large es big. Fix: medium o distil
OOM en GPU	Fix: chunk_length_s=30 para procesar por c
Hallucinations en silencios	Whisper a veces "inventa" texto en silenci
Idioma detectado mal	Fix: language='es' explícito.

Preguntas frecuentes

Whisper vs Google STT / Azure?

Whisper es gratis y open. Calidad comparable o mejor en muchos idiomas. Google/Azure mejor en real-time streaming.

Distil-Whisper cuándo?

Cuando latencia importa o tenés CPU. Calidad ~1-2 WER points peor que full Whisper-large. Worth it.

Diarization (multiple speakers)?

Whisper NO la hace. Combinar con pyannote.audio.

Real-time streaming?

Whisper es batch (audio completo). Para streaming: faster-whisper con VAD, o insanely-fast-whisper (BetterTransformer + Flash Attention).

Hallucination prevention?

condition_on_previous_text=False, temperature alta cuando no hay confianza, VAD para skip silencios.

Referencias

- Radford et al. (2022), Robust Speech Recognition via Large-Scale Weak Supervision, OpenAI.
- Whisper repo.
- Distil-Whisper.
- Insanely Fast Whisper.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 148 — Clase 148 — LLMs aplicados: fine-tuning, prompting (+ LoRA / QLoRA, DPO, vLLM)

> *Parte: 2 — Deep Learning · Fuente: docs HuggingFace PEFT + TRL + vLLM + papers LoRA (Hu et al. 2021), QLoRA (Dettmers et al. 2023), DPO (Rafailov et al. 2023). Duración estimada: 110 min.*

Objetivo

Manejar Large Language Models (Llama 3, Mistral, Qwen, etc.) en flujos reales: prompting técnico (zero-shot, few-shot, chain-of-thought), fine-tuning eficiente con LoRA / QLoRA (entrenar solo 0.1-1 % de parámetros), alineamiento con DPO (preferencias, en lugar de RLHF clásico), e inference de producción con vLLM (continuous batching, PagedAttention).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diseñar prompts con system prompts, few-shot examples y chain-of-thought.
- Aplicar LoRA con peft library: agregar adapters chicos a un LLM grande, entrenar solo ~0.5 % de params.
- Aplicar QLoRA (cuantización 4-bit) para fine-tunear Llama 7B en una sola GPU de 24 GB.
- Alinear con preferencias usando DPO (trl library) — más simple y estable que RLHF.
- Servir un modelo con vLLM y medir throughput.

Temas

- LLM stack en 2026: base → SFT (Supervised Fine-Tuning) → DPO/RLHF → inference optimizada.

- Prompting técnico: structure, few-shot, chain-of-thought, function calling.
- PEFT (Parameter-Efficient Fine-Tuning): LoRA, QLoRA, adapters.
- DPO vs RLHF.
- vLLM: continuous batching, PagedAttention, OpenAI-compatible API.
- Evaluación: MMLU, HumanEval, MT-Bench, LMSys Arena.

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 128a — LoRA / QLoRA: fine-tuning eficiente de LLMs
- Clase 128b — DPO y RLHF: alineamiento de LLMs
- Clase 128c — vLLM y TGI: serving de LLMs en producción

Definiciones y características

- Prompting: arte de estructurar el input para que el modelo dé buenos outputs.
- Few-shot: incluir 2-5 ejemplos del task en el prompt.
- Chain-of-thought (CoT): pedirle al modelo que "piense paso a paso" — mejora dramática en tareas razonamiento.
- System prompt: instrucciones generales antes del diálogo (rol, restricciones).
- PEFT: técnica de fine-tuning con pocos parámetros.
- LoRA: rank-r decomposition para los deltas.
- QLoRA: LoRA + 4-bit quantization del base.
- DPO: loss directa sobre preferencias, sin reward model.
- vLLM: framework de inference optimizada open-source.
- KV cache: cache de K, V de tokens previos, crítico para inference autoregresiva.

Dataset / recursos

- HuggingFace Hub para modelos.
- Datasets de instrucción: Alpaca, ShareGPT, OpenOrca.
- Datasets de preferencia: Anthropic HH-RLHF, UltraFeedback.
- Librerías: transformers, peft, trl, bitsandbytes, vllm.

Ejercicios

1. Prompt engineering: para una tarea (e.g., clasificación de quejas), iterar 5 versiones de prompt (zero-shot, few-shot, CoT, etc.). Medir accuracy en un set chico.
2. LoRA SFT: fine-tunear un Mistral 7B Instruct con LoRA sobre un dataset propio chico (1k-10k ejemplos).
3. QLoRA: el mismo experimento pero con quantization 4-bit. Comparar memoria y velocidad.
4. DPO: con un dataset de preferencias mínimo (e.g., 500 pairs), aplicar DPO sobre el modelo SFT.
5. vLLM serving: levantar vLLM con el modelo + LoRA adapter, medir throughput vs HF pipeline.

Homework verificable

Fine-tunear un LLM open con LoRA para una tarea propia:

1. Elegir Llama 3 8B Instruct o Mistral 7B Instruct.
2. Dataset propio: 500-2000 pares (instrucción, respuesta).
3. QLoRA config: r=16, target=['q_proj','k_proj','v_proj','o_proj'], 4-bit.
4. Entrenar 3 épocas; evaluar perplexity en val.
5. Servir con vLLM y testear 10 ejemplos.

Criterio de aceptación: perplexity en val mejora vs base; el modelo responde apropiadamente al dominio para los 10 ejemplos test.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
RuntimeError: CUDA out of memory con Llama	Sin quantization. Fix: QLoRA con bnb_4bit.
LoRA con target_modules='all-linear' lento	Demasiados adapters. Fix: solo q/v en atte
Modelo SFT sobre dataset chico pierde capa	Catastrophic forgetting parcial. Fix: subi
DPO sin ref_model actualizado	El ref model debe ser SFT init. Fix: usar
vLLM no carga LoRA adapter	Hay que mergear LoRA + base primero o usar

Preguntas frecuentes

¿Open-source o API (GPT-4, Claude)?

API para prototipos, casos non-críticos. Self-host con open-source cuando importan: privacidad, costo a escala, control total. Ambos coexisten.

¿LoRA o full fine-tune?

LoRA en 99 % de los casos. Full fine-tune solo si tenés cluster grande y querés cambiar significativamente la base.

¿DPO o RLHF?

DPO por default — más simple y casi tan bueno. RLHF para tareas muy específicas donde necesitás un reward model fino.

¿vLLM o TGI?

vLLM ligeramente más rápido en benchmarks. TGI (HuggingFace) más integrado con HF Hub. Ambos buenos.

¿Cuánto cuesta entrenar un LLM desde cero?

Llama 3 70B: estimado $10M-50M en compute. Para custom: forget about it — fine-tunea uno existente.

Referencias

- Hu et al. (2021), LoRA: Low-Rank Adaptation of Large Language Models, ICLR.
- Dettmers et al. (2023), QLoRA: Efficient Finetuning of Quantized LLMs, NeurIPS.
- Rafailov et al. (2023), Direct Preference Optimization, NeurIPS.
- Kwon et al. (2023), Efficient Memory Management for LLM Serving with PagedAttention, SOSP.
- PEFT docs, TRL docs, vLLM docs.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 149 — Clase 149 — LoRA / QLoRA: fine-tuning eficiente de LLMs

Parte: 2 — Deep Learning · Fuente: Hu et al. (2021) LoRA + Dettmers et al. (2023) QLoRA. Duración estimada: 100 min.

Objetivo

Hacer fine-tuning de LLMs (Llama 3, Mistral, Qwen 2) en una GPU consumer con LoRA y QLoRA. Cubrir hyperparámetros (r, alpha, dropout, target_modules), inspección de trainable params, merge LoRA con base, y deployment.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar LoRA con peft.LoraConfig y get_peft_model.
- Configurar QLoRA con bitsandbytes 4-bit + BitsAndBytesConfig(load_in_4bit=True, bnb_4bit_quant_type='nf4').
- Elegir rank r (típico 8-64), alpha (típico 2×r), target_modules.
- Mergear el LoRA adapter con el base para deploy: model.merge_and_unload().
- Calcular trainable params vs total: ~0.1-1 % es el ratio típico.

Temas

- LoRA matemáticamente: W' = W + (B·A)·alpha/r.
- QLoRA: NF4 quantization + double quant + paged optimizers.
- target_modules: ['q_proj', 'k_proj', 'v_proj', 'o_proj'] clásico; o all-linear.
- Save / load: solo el adapter (~10-50 MB) en lugar del full model.
- Merge + serve vs separated adapter.

Definiciones y características

- LoRA: matrices rank-r entrenables agregadas a Linear layers.
- r (rank): 8-16 para tareas chicas; 32-64 para datos grandes/cambios fuertes.
- alpha: scaling. Convención: alpha = 2r.
- NF4: NormalFloat 4-bit, optimizado para weights de redes pre-trained.
- Double quantization: cuantizar los scales también; ahorra ~0.4 bits/param.
- merge_and_unload(): devuelve un modelo "normal" con LoRA aplicado.

Dataset / recursos

- HuggingFace dataset de instrucción (Alpaca, Dolly, OpenAssistant).
- Modelo base: Llama 3 8B Instruct, Mistral 7B Instruct, Qwen 2 7B Instruct.
- Librerías: transformers, peft, bitsandbytes, accelerate, trl.

Ejercicios

1. LoRA básico: cargar Mistral 7B Instruct sin quantization; aplicar LoRA r=16. Inspeccionar trainable params (~0.5 %).
2. QLoRA: ahora con load_in_4bit=True. Verificar uso VRAM (~6 GB).
3. Train con TRL: SFTTrainer sobre 500 ejemplos custom. ~30 min en GPU consumer.
4. Inference con adapter: cargar base + LoRA y predecir.
5. Merge: model.merge_and_unload() → save como modelo normal.

Homework verificable

Fine-tune Mistral 7B Instruct con QLoRA para un dominio propio:

1. Dataset: 200-500 pares (instrucción, respuesta).
2. QLoRA r=16, target=all-linear, 3 épocas.
3. Inference con el adapter: 10 ejemplos test.
4. Reportar mejora cualitativa vs base.

Criterio de aceptación: respuestas más alineadas al dominio que el base; uso de VRAM < 16 GB durante training.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
OOM en Llama 7B + Adam	Sin quantization. Fix: QLoRA + paged_adamw
LoRA target_modules='all-linear' lento	Demasiados adapters. Fix: solo q/v y luego
Trainable params 0	Olvidaste get_peft_model. Fix: aplicarlo a
Resultados peor que base	LR mal, target_modules incompleto, o pocos
merge_and_unload modifica el base	Por design, en place. Fix: si necesitás ba

Preguntas frecuentes

rank r óptimo?

Empieza con 16. Tareas pequeñas: 8. Datos grandes / cambios sustanciales: 32-64.

¿LoRA o QLoRA?

QLoRA si la GPU no aguanta. Sino LoRA, ligeramente mejor calidad final.

Multi-adapter?

Sí — entrenar varios adapters (uno por tarea/idioma) y switchear en inference con model.set_adapter('name').

¿LoRA en visión / difusión?

Sí, mismo concepto. diffusers tiene LoRA support para SDXL fine-tuning.

DoRA, AdaLoRA?

Variantes: DoRA (weight-decomposed) y AdaLoRA (rank adaptativo). Mejoras marginales. LoRA "vanilla" sigue siendo default.

Referencias

- Hu et al. (2021), LoRA, ICLR.
- Dettmers et al. (2023), QLoRA, NeurIPS.
- PEFT docs.
- TRL docs.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 150 — Clase 150 — DPO y RLHF: alineamiento de LLMs

> *Parte: 2 — Deep Learning · Fuente: Ouyang et al. (2022) InstructGPT/RLHF + Rafailov et al. (2023) DPO + papers IPO/KTO/ORPO. Duración estimada: 95 min.*

Objetivo

Alinear LLMs con preferencias humanas (helpful, harmless, honest). Cubrir RLHF clásico (SFT → Reward Model → PPO, complejo) y DPO (Direct Preference Optimization, moderno y simple). Conocer variantes 2023-2024: IPO, KTO, ORPO.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar el pipeline RLHF de 3 etapas (SFT, RM, PPO).
- Aplicar DPO con trl.DPOTrainer sobre un dataset de preferencias.
- Diferenciar DPO (single-step) de KTO (no requiere pairs) y ORPO (combina SFT + alineamiento).
- Crear un dataset de preferencias: (prompt, chosen, rejected).
- Evaluar alineamiento con MT-Bench, AlpacaEval, o LLM-as-judge.

Temas

- Por qué alineamiento: LLM pretrained → genera pero no sigue instrucciones bien ni evita harm.
- SFT (Supervised Fine-Tuning): instruction tuning.
- Reward Model: regression entrenada con human preferences.
- PPO: optimizar el LLM contra el RM.
- DPO: derivación matemática que elimina el RM.
- IPO: variante con identity link, más estable.
- KTO: solo chosen o rejected (no necesita pairs).
- ORPO: alineamiento desde SFT directamente.

Definiciones y características

- SFT: training supervisado con (prompt, response_humana).
- Reward Model: predicen r(prompt, response).
- Bradley-Terry: modelo probabilístico P(A > B) = σ(r(A) - r(B)).
- DPO loss: -log σ(β·(log π(chosen)/π_ref(chosen) - log π(rejected)/π_ref(rejected))).
- β (DPO): control del KL respecto al ref model. 0.1-0.5 típico.
- Reference model: copia frozen del SFT, usada para regularizar.

Dataset / recursos

- Anthropic HH-RLHF, UltraFeedback, Argilla DPO datasets.
- Modelo base: SFT propio (clase 128a) o Mistral 7B Instruct.
- Librerías: transformers, trl, peft, bitsandbytes.

Ejercicios

1. Dataset de preferencias: cargar Anthropic/hh-rlhf. Inspeccionar chosen y rejected.
2. DPO con TRL: DPOTrainer(model, ref_model, ...) con LoRA encima. Train 1 época.
3. Eval pre/post: generar respuestas a 20 prompts antes y después; comparar manualmente.
4. β sensitivity: probar β {0.1, 0.3, 1.0}. β alto → menos cambio; β bajo → más agresivo.
5. KTO: dataset con solo chosen (no pairs). Aplicar KTOTrainer.

Homework verificable

DPO sobre un dominio propio:

1. Dataset de 200-500 pares (puede ser sintético con LLM como judge).
2. Base: modelo SFT propio (de 128a).
3. DPO con LoRA, β=0.1.
4. Comparar 20 respuestas pre/post; evaluar con LLM-as-judge (e.g., Claude).

Criterio de aceptación: ≥ 60 % de los outputs post-DPO son juzgados mejores que pre.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
DPO degrada calidad	β muy bajo o demasiadas épocas. Fix: β=0.1
Reward hacking en RLHF	Modelo aprende a hacer trampas al RM. Fix:
ref_model consume mucha VRAM	Copia frozen del modelo. Fix: usar ref_mod
Dataset de pairs muy ruidoso	Annotators inconsistentes. Fix: filtering,
Resultados malos sin SFT previo	DPO asume modelo razonable. Fix: SFT prime

Preguntas frecuentes

DPO o RLHF clásico?

DPO por default 2024+ — más simple, casi igual calidad. RLHF si tenés team y reward model bueno (e.g., GPT-4 / Claude para producción).

¿IPO, KTO, ORPO cuál?

DPO sigue siendo default. IPO si DPO inestable. KTO si solo tenés chosen. ORPO combina SFT+pref → más eficiente, en alza.

¿Dataset de cuántos pairs?

10k-50k para alineamiento serio. 500-2000 para experimentos.

Evaluación cómo?

LLM-as-judge (GPT-4/Claude evalúa pares), MT-Bench, AlpacaEval. Human eval para gold standard.

¿DPO en LLMs > 70B?

Sí, con FSDP / DeepSpeed. Trabajo "expensive" pero factible.

Referencias

- Ouyang et al. (2022), Training language models to follow instructions with human feedback (InstructGPT), NeurIPS.
- Rafailov et al. (2023), Direct Preference Optimization, NeurIPS.
- Ethayarajh et al. (2024), KTO, NeurIPS.
- Hong et al. (2024), ORPO.
- TRL docs.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 151 — Clase 151 — vLLM y TGI: serving de LLMs en producción

> *Parte: 2 — Deep Learning · Fuente: Kwon et al. (2023) vLLM + HuggingFace TGI docs. Duración estimada: 80 min.*

Objetivo

Servir LLMs eficientemente en producción con vLLM (Berkeley) o TGI (HuggingFace). Cubrir: PagedAttention, continuous batching, prefill/decode, quantization (AWQ, GPTQ, FP8), structured output (JSON, function calling), streaming, OpenAI-compatible API.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Levantar vLLM serving: python -m vllm.entrypoints.openai.api_server --model X.
- Hacer requests con OpenAI client apuntando a vLLM.
- Aplicar continuous batching y entender ganancia de throughput.
- Servir modelos cuantizados (AWQ, GPTQ, FP8) para reducir VRAM.
- Activar structured outputs con guided_json (JSON schema enforced).

Temas

- KV cache: por qué crece con secuencia.
- PagedAttention (vLLM): page table como OS → no fragmentación.
- Continuous batching: nuevos requests entran sin esperar al batch.
- Prefill (compute KV) vs decode (1 token/step).
- Speculative decoding: predecir N tokens, verificar con modelo grande.
- Quantization: FP8 (H100), AWQ/GPTQ (4-bit weight-only).

Definiciones y características

- vLLM: framework serving open-source. PyTorch-based. ~5-20× throughput vs HF pipeline.
- TGI (Text Generation Inference): HF Hub-integrado. Similar a vLLM.
- PagedAttention: KV cache paginado.
- AWQ / GPTQ: 4-bit weight-only quantization, mantiene calidad ~98 %.
- FP8: 8-bit float native en H100. Mejor que int8 numéricamente.
- Speculative decoding: 2-3× speedup en decode con modelo draft.

Dataset / recursos

- Modelo: Mistral 7B Instruct, Llama 3 8B Instruct, o uno cuantizado AWQ.
- Librerías: vllm, transformers, openai (client).

Ejercicios

1. vLLM básico: python -m vllm.entrypoints.openai.api_server --model mistralai/Mistral-7B-Instruct-v0.2. Cliente OpenAI.
2. Continuous batching benchmark: 100 requests paralelos vs 1 a la vez. Comparar throughput.
3. AWQ quantization: cargar TheBloke/Mistral-7B-Instruct-v0.2-AWQ. VRAM ~5 GB vs 14 GB fp16.
4. Structured JSON output: extra_body={'guided_json': {schema}} → forced JSON valid.
5. TGI: docker run --gpus all ghcr.io/huggingface/text-generation-inference:latest --model-id X.

Homework verificable

Servir Mistral 7B Instruct AWQ con vLLM + cliente OpenAI:

1. Levantar server vLLM.
2. 50 prompts batch en paralelo desde cliente.
3. Medir throughput (tokens/sec total).
4. Comparar contra transformers.pipeline generate.
5. Activar guided_json para una tarea específica.

Criterio de aceptación: vLLM ≥ 5× throughput vs HF pipeline; structured output válido al 100 %.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
OOM al cargar	Modelo + KV cache no entra. Fix: quantizat
Throughput bajo en una sola request	Con batch=1, vLLM es similar a HF. Fix: pa
LoRA adapter no carga	Fix: vLLM soporta LoRA con --enable-lora -
Structured output formato inválido	guided_json schema mal. Fix: JSON Schema v
TGI lento	Versión vieja. Fix: usar imagen latest.

Preguntas frecuentes

vLLM vs TGI?

vLLM ligeramente más rápido en benchmarks; mejor LoRA support. TGI más integrado con HF Hub. Ambos buenos.

Ollama / llama.cpp?

llama.cpp / Ollama: CPU-friendly, GGUF format. Para desktop, local, low traffic. vLLM/TGI para servers con GPU.

Estructura JSON garantizada?

Sí con guided_json (vLLM uses outlines library). Mucho más confiable que prompt engineering.

Speculative decoding?

--speculative-model X --num-speculative-tokens 5. 2-3× decode speedup con calidad idéntica.

Multi-GPU?

--tensor-parallel-size N para shard del modelo en N GPUs.

Referencias

- Kwon et al. (2023), Efficient Memory Management for LLM Serving with PagedAttention, SOSP.
- vLLM docs.
- TGI docs.
- Outlines (structured output).

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 152 — Clase 152 — RAG básico y embeddings (+ hybrid search, re-ranking, MCP)

> *Parte: 2 — Deep Learning · Fuente: Lewis et al. (2020) RAG paper + LlamaIndex / LangChain docs + Anthropic MCP spec. Duración estimada: 100 min.*

Objetivo

Construir un sistema RAG (Retrieval-Augmented Generation) — pipeline que enriquece a un LLM con conocimiento externo: documentos propios, base de datos, web. Pipeline: embedding los docs → almacenar en vector DB → al query, hacer retrieval de los k más relevantes → inyectar como contexto al LLM → respuesta basada en docs. Conocer mejoras modernas: hybrid search (denso + sparse), cross-encoder re-ranking, y el Model Context Protocol (MCP) que estandariza la conexión LLM-herramientas.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Generar embeddings de texto con sentence-transformers o modelos HF.
- Almacenar y buscar embeddings con FAISS, Chroma, Pinecone, Qdrant, o pgvector.
- Construir el flujo query → embed → top-k → context → LLM.
- Aplicar hybrid search: combinar BM25 (sparse) con embeddings (dense) → mejor recall.
- Aplicar cross-encoder re-ranking sobre los top-100 retornados para promover los top-10 reales.
- Reconocer el MCP como protocolo abierto para conectar LLMs a herramientas (filesystems, DBs, APIs).

Temas

- Por qué RAG: LLMs no saben de documentos privados; entrenar fine-tune no escala para conocimiento dinámico.
- Embeddings: vectores de dimensión ~768-1536.
- Vector DBs: FAISS (in-memory), Chroma (local), Qdrant/Weaviate (server), pgvector (Postgres extension).
- Chunking: dividir docs en piezas de ~200-1000 tokens.
- Top-k retrieval + context window.
- Complemento moderno: hybrid search, cross-encoder rerank, MCP.

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 129a — MCP (Model Context Protocol)
- Clase 129b — Agentes: tool use, ReAct, multi-agent
- Clase 129c — LLM Evaluation: MMLU, MT-Bench, LLM-as-judge

Definiciones y características

- Embedding model: red que mapea texto a vector denso. all-MiniLM-L6-v2 (small, free), text-embedding-3-large (OpenAI, mejor calidad).
- Vector DB: base optimizada para similarity search aproximada (HNSW, IVF).
- Chunking: dividir documentos largos. Trade-off: chunks chicos → contexto perdido; grandes → ruido.
- Top-k: cuántos docs retorna el retriever. Típico 5-20.
- Reranker: modelo que re-puntúa pares (query, doc).
- MCP: protocolo standardizado de Anthropic para tool use.

- RAG-Fusion: variante que genera N variantes del query y agrega resultados.

Dataset / recursos

- Cualquier corpus de docs propios (PDFs, markdown, HTML).
- Wikipedia dump para experimentos.
- Librerías: sentence-transformers, chromadb / faiss-cpu, rank_bm25, langchain / llama-index, mcp.

Ejercicios

1. Embed + index: con sentence-transformers/all-MiniLM-L6-v2, embeber 100 párrafos y guardarlos en Chroma. Query y top-5.
2. BM25 baseline: con rank_bm25, query y comparar resultados vs dense.
3. Hybrid (RRF): combinar ambos. Verificar que hybrid > dense > BM25 sola en queries técnicas.
4. Cross-encoder rerank: tomar top-50 del paso 3, rerankar con cross-encoder/ms-marco-MiniLM-L-6-v2. Comparar nDCG.
5. RAG con LLM: top-5 → contexto + query → Claude o GPT → respuesta con citas.

Homework verificable

Mini RAG sobre 100-1000 documentos propios:

1. Chunking + embedding + indexado en Chroma.
2. Pipeline retrieval (dense), generate (LLM via API o vLLM local).
3. 10 queries de prueba.
4. Evaluar manualmente: ¿cuántas respuestas correctas con citas válidas?
5. Bonus: agregar BM25 + cross-encoder rerank y medir mejora.

Criterio de aceptación: al menos 7/10 respuestas correctas con citas válidas; hybrid + rerank mejora el ratio.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Retrieval devuelve docs irrelevantes	Chunks muy grandes o embeddings malos. Fix
LLM alucina aunque hay contexto	Prompt no instruye a "responder solo con e
Recall bajo en queries con nombres propios	Dense pierde. Fix: hybrid con BM25.
Vector DB lento con millones de docs	Sin índice ANN. Fix: HNSW (default en Chro
Citas falsas (LLM inventa números)	El modelo no respeta formato. Fix: usar fu

Preguntas frecuentes

¿RAG o fine-tuning?

RAG para conocimiento dinámico, citable, separable del modelo. Fine-tune para skills (formato de respuesta, tono). A menudo ambos: fine-tune para how, RAG para what.

¿Cuál vector DB?

- Prototipo local: Chroma o FAISS.
- Producción small: Qdrant (Rust, rápido).
- Postgres existente: pgvector.
- Enterprise: Pinecone (managed, sin maintenance).

¿Embeddings open o API?

Open: bge-large-en-v1.5, Nomic-embed-text-v1.5. API: text-embedding-3-large (OpenAI), voyage-3 (Voyage

AI). API es mejor pero costo recurrente.

¿Cómo evalúo el RAG?

Métricas: Recall@k (¿el doc correcto está entre top-k?), MRR, nDCG. Para end-to-end: RAGAS library, LLM-as-judge.

¿MCP en producción?

Sí, ya. Anthropic y comunidad ofrecen MCP servers para muchas tools comunes. Adopt earlier para ganar portabilidad cross-LLM.

Referencias

- Lewis et al. (2020), Retrieval-Augmented Generation, NeurIPS — paper original.
- Reimers & Gurevych (2019), Sentence-BERT, EMNLP.
- Robertson & Zaragoza (2009), BM25 and Beyond.
- Anthropic (2024), Model Context Protocol — <https://modelcontextprotocol.io/>.
- LlamaIndex docs, LangChain docs.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 153 — Clase 153 — MCP (Model Context Protocol): herramientas y datos para LLMs

> *Parte: 2 — Deep Learning · Fuente: MCP spec (Anthropic 2024). Duración estimada: 80 min.*

Objetivo

Aprender MCP (Model Context Protocol) — estándar abierto publicado por Anthropic en noviembre 2024 que define cómo un LLM se conecta a herramientas externas (filesystems, databases, APIs, search engines, etc.). Antes de MCP, cada framework (LangChain, LlamaIndex, OpenAI plugins) tenía API propia. MCP unifica → portabilidad entre LLMs y clients.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar la arquitectura MCP: client, server, resources, tools, prompts.
- Conectar MCP servers existentes a Claude Desktop, Cursor, Zed.
- Escribir un MCP server propio en Python (con fastmcp).
- Diferenciar MCP de tool use clásico de OpenAI / LangChain.
- Usar MCP servers populares: filesystem, postgres, git, slack, brave-search.

Temas

- Cliente (LLM app) vs Server (provee tools/resources/prompts).
- Transport: stdio (local) y SSE (network).
- Resources: read-only data (archivos, DB rows).
- Tools: funciones invocables (search, write).

- Prompts: templates reusables.
- Discovery: el client descubre dinámicamente qué hay disponible.

Definiciones y características

- MCP server: programa que expone resources/tools/prompts via JSON-RPC.
- MCP client: typically un LLM app (Claude Desktop, Cursor, IDE plugins).
- Resource URI: e.g., file:///path/to/x, postgres://db/table.
- Tool schema: JSON Schema describing inputs/outputs.
- fastmcp: librería Python para escribir servers MCP rápido.

Dataset / recursos

- MCP servers oficiales.
- Librerías: mcp (Python SDK), fastmcp.

Ejercicios

1. Conectar server existente: instalar mcp-server-filesystem en Claude Desktop. Hacer queries sobre archivos del file system.
2. Server propio: con fastmcp, exponer un tool search_docs(query) sobre un corpus local.
3. Resource: exponer archivos .md como resources lectura.
4. Prompt template: definir summarize(file_uri) como prompt reusable.
5. Multi-server: conectar 2-3 servers simultáneos a Claude Desktop; usar conjuntamente.

Homework verificable

MCP server propio para RAG sobre documentación:

1. fastmcp con tool search(query) que usa el RAG de clase 129.
2. Resource list_documents() que lista archivos indexados.
3. Conectar a Claude Desktop; usarlo en una conversación.
4. Comparar UX vs hacer RAG manual en código.

Criterio de aceptación: el LLM cliente puede llamar al search tool naturalmente y trae resultados relevantes.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Server no aparece en client	Config mal en ~/.config/claude/.... Fix: v
Tool schema rejected	JSON Schema inválido. Fix: validar con un
stdio server no inicia	Path al executable mal. Fix: paths absolut
Server lento	Llamadas síncronas. Fix: usar async def to
Cliente no encuentra resource	URI mal formado. Fix: schema correcto.

Preguntas frecuentes

¿MCP vs LangChain tools?

LangChain tools son Python objects acoplados al framework. MCP es un protocol — funciona con cualquier LLM client que lo soporte. Más portable.

¿OpenAI plugins / GPTs?

Más cerrados, especificos a OpenAI. MCP open + multi-vendor.

Servers existentes utiles?

filesystem, git, postgres, sqlite, slack, brave-search, fetch (HTTP), memory, github, gdrive — muchos en el repo oficial.

Seguridad?

Cada server corre con permisos del usuario. Cuidado con qué exponés. Auditar antes de instalar de terceros.

¿OpenAI agrega MCP?

Anthropic open-sourceó el spec; otros vendors lo están adoptando (Microsoft Copilot Studio, etc.). Está volviéndose estándar.

Referencias

- MCP spec.
- MCP servers official repo.
- fastmcp.
- Anthropic blog (Nov 2024), Introducing the Model Context Protocol.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 154 — Clase 154 — Agentes: tool use, ReAct, multi-agent

> *Parte: 2 — Deep Learning · Fuente: Yao et al. (2023) ReAct + Schick et al. (2023) Toolformer + Anthropic agent patterns (2024). Duración estimada: 95 min.*

Objetivo

Construir agentes con LLMs: el LLM planifica, invoca tools (funciones, MCP servers, APIs), observa los resultados y decide el siguiente paso. Cubrir el paradigma ReAct (Reasoning + Acting), tool use moderno (function calling structured), y patrones multi-agent (workflows, swarms, supervisores).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Definir tools con JSON schema; pasar a la API del LLM.
- Implementar loop ReAct manual: prompt → LLM → tool call → execute → observation → prompt.
- Usar LangGraph o AutoGen o CrewAI para orquestar multi-agent.
- Diseñar patrones: workflow (lineal), router (decide ruta), evaluator-optimizer (loop con auto-crítica).
- Reconocer cuándo NO usar agentes (tareas determinísticas → workflow simple; agentes solo si hace falta razonamiento dinámico).

Temas

- Tool use con OpenAI/Anthropic function calling.
- ReAct prompt template.
- Loops: while LLM produces tool_call, execute and feed observation.

- LangGraph: state machines para agentes.
- AutoGen / CrewAI: multi-agent frameworks.
- Patrones (Anthropic 2024): prompt chaining, routing, parallelization, orchestrator-workers, evaluator-optimizer.
- Token cost: agentes con loops pueden gastar mucho.

Definiciones y características

- Tool: función con schema + descripción que el LLM puede invocar.
- Function calling: el LLM devuelve structured {tool_name, arguments} en lugar de texto.
- ReAct: prompt template Thought → Action → Observation → Thought → ....
- Agent: LLM en loop con tools y memoria de pasos previos.
- Supervisor pattern: 1 LLM coordina varios sub-agents.
- Token bound: agentes pueden hacer N iteraciones; siempre poner max_iterations.

Dataset / recursos

- Anthropic Claude API o OpenAI API (function calling).
- Librerías: anthropic, openai, langgraph, crewai, autogen.

Ejercicios

1. Tool básico: definir weather(city) y search(query) tools en Claude API. Pedirle que use ambos.
2. ReAct loop manual: implementar el loop sin librería, en Python puro.
3. LangGraph: definir un grafo: router → tool → evaluator → end. Compilar y ejecutar.
4. Multi-agent (CrewAI): 3 agents (researcher, writer, editor) para producir un blog post.
5. Evaluator-optimizer: agent que escribe + agent que critica + loop hasta approved=True.

Homework verificable

Agente de investigación con tools:

1. Tools: web_search, fetch_url, extract_text, save_note.
2. Prompt: "investigá X y produce un resumen de 200 palabras citando fuentes".
3. Loop ReAct hasta produce el resumen.
4. Reportar # de iteraciones, # de tool calls, costo en tokens.

Criterio de aceptación: resumen producido con ≥ 3 fuentes citadas; max_iterations no excedido.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Loop infinito	LLM repite mismas tool calls. Fix: max_ite
Tool falla, LLM no maneja	Fix: pasar el error como observation.
Costo explota	Agentes son caros. Fix: usar Haiku/GPT-min
Tool schema mal	LLM no llama. Fix: descriptions claras, sc
Output no estructurado	Fix: forzar JSON schema en respuesta final

Preguntas frecuentes

Agentes vs workflows simples?

Workflow simple si la secuencia es fija. Agente cuando el path depende del input. Anthropic recomienda: empieza simple, agregá agentic complexity solo cuando justifica.

LangGraph vs CrewAI vs AutoGen?

- LangGraph: state machines explícitos, control fino.
- CrewAI: roles + delegation, fácil para multi-agent estilo "team".
- AutoGen: conversaciones entre agents, research-oriented.

MCP en agentes?

Sí — los MCP servers son una fuente de tools estándar. Combinar MCP + agente.

Eval de agentes?

Difícil. AgentBench, τ-Bench, SWE-Bench (coding). Para custom: LLM-as-judge + human spot check.

Memoria persistent?

Vector DB + tool recall(query). O frameworks como mem0.

Referencias

- Yao et al. (2023), ReAct: Synergizing Reasoning and Acting, ICLR.
- Schick et al. (2023), Toolformer, NeurIPS.
- Anthropic (2024), Building Effective Agents — patterns guide.
- LangGraph docs, CrewAI, AutoGen.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 155 — Clase 155 — LLM Evaluation: MMLU, MT-Bench, LLM-as-judge, evals propios

> *Parte: 2 — Deep Learning · Fuente: Hendrycks et al. (2021) MMLU + Zheng et al. (2023) MT-Bench + LMSys Arena. Duración estimada: 85 min.*

Objetivo

Evaluar LLMs (propios o terceros) con rigor: benchmarks estándar (MMLU, HumanEval, GSM8K, MT-Bench, LMSys Arena), LLM-as-judge para casos open-ended, y evals propios específicos al dominio. Reconocer las trampas (data contamination, reward hacking, leaderboard hacking).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Correr MMLU con lm-evaluation-harness sobre un modelo propio.
- Implementar LLM-as-judge con (prompt, response_A, response_B) → "cuál es mejor".
- Diseñar evals custom: cobertura por tema, casos edge, regresiones.
- Diferenciar classification metrics (accuracy on MCQs) de generation metrics (BLEU, ROUGE, BERTScore, LLM-judge).
- Reconocer data contamination (test set en pretraining) y leaderboard hacking.

Temas

- MMLU: 57 dominios, multiple choice. Standard 2020-2023.
- HumanEval: 164 problemas Python codegen.
- GSM8K: math word problems.
- MT-Bench: 80 multi-turn questions evaluadas por GPT-4 judge.
- LMSys Arena: head-to-head humanos. Standard moderno (ELO ranking).
- LLM-as-judge: usar GPT-4/Claude como evaluador.
- Custom evals: críticos para producción.

Definiciones y características

- MMLU: 15k MCQs, 57 categorías. Score 0-100.
- MT-Bench: 80 prompts multi-turno; GPT-4 score 1-10 cada respuesta.
- LMSys Arena: humanos votan A vs B; ELO global.
- Pass@k: codegen — % de problemas resueltos en k intentos.
- LLM-as-judge: criterios estructurados, comparison pairs.
- Data contamination: el modelo memorizó el test → metrics infladas.

Dataset / recursos

- HuggingFace: lm-eval-harness, HELM, lighteval.
- Modelo a evaluar: cualquier LLM open o API.
- Librerías: lm-evaluation-harness, inspect_ai, promptfoo.

Ejercicios

1. MMLU con lm-eval-harness: lm_eval --model hf --model_args pretrained=mistralai/Mistral-7B-v0.1 --tasks mmlu --num_fewshot 5. Reportar score.
2. HumanEval: code generation, pass@1.
3. MT-Bench: usar GPT-4 / Claude como judge. Reportar score promedio.
4. LLM-as-judge propio: 20 pairs (model_A vs model_B); judge devuelve A/B/tie + reasoning.
5. Custom eval: 50 prompts específicos a tu use case + criterios de aceptación.

Homework verificable

Evaluar 2 modelos (e.g., Mistral 7B vs Llama 3 8B) en una tarea propia:

1. 30 prompts específicos al dominio.
2. Generar respuestas con ambos.
3. LLM-as-judge (Claude o GPT-4) comparando.
4. Reportar win rate de cada uno.

Criterio de aceptación: judge entrega resultado consistente (inter-rater agreement > 0.7 entre 2 ejecuciones).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
MMLU score muy alto sospechoso	Posible data contamination. Fix: verificar
LLM-as-judge sesgado por longitud	Tiende a preferir respuestas largas. Fix:
LLM-judge sesgado por orden A/B	Si A siempre se evalúa primero, sesgo. Fix
Eval gameable	Optimizar en el test → degrada producción.
GPU OOM con lm-eval	Modelo grande. Fix: --batch_size auto:1 y

Preguntas frecuentes

MMLU sigue siendo válido en 2026?

Sí pero saturado — top modelos > 88 %. Mejores: MMLU-Pro, GPQA, BBH, MATH.

LMSys Arena reliable?

Sí, gold standard humano. Caro (necesita usuarios). Para producción usar como ground truth.

LLM-as-judge confiable?

GPT-4 / Claude como judge correlatean ~85 % con humanos en MT-Bench. Bias conocidos (length, position). Mitigar con prompts cuidadosos.

Evals para agentes?

SWE-Bench (coding), τ-Bench (tool use), AgentBench. Custom para tu workflow.

Eval continuo en producción?

Sample logs → LLM-as-judge daily → alertar si win-rate baja vs baseline.

Referencias

- Hendrycks et al. (2021), MMLU, ICLR.
- Zheng et al. (2023), Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena, NeurIPS.
- LM Evaluation Harness.
- LMSys Arena.
- Inspect AI — UK AISI eval framework.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 156 — Clase 156 — Autoencoders: undercomplete, stacked, denoising, sparse

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 17 § Autoencoders, GANs, and Diffusion Models. Duración estimada: 70 min.*

Objetivo

Entender autoencoders — red Encoder → bottleneck → Decoder entrenada a reconstruir su input. Variantes que cubrimos: undercomplete (dim_latent < dim_input, fuerza compresión), stacked (deep), denoising (input ruidoso → output limpio), sparse (penaliza activaciones latentes). Saber qué problemas resuelven (compresión, anomaly detection, pretraining) y cuándo VAEs/GANs/Diffusion los superan en generación.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Construir un undercomplete AE con MLP/CNN y entrenarlo con MSE.
- Diferenciar latent_dim < input_dim (compresión real) de latent_dim >> input_dim con regularización (sparse).
- Implementar Denoising AE: input x + noise, target x.

- Aplicar AE como anomaly detector: alta reconstruction error → anomalía.
- Reconocer que autoencoders no son buenos generadores (latent space irregular) → motivó VAE (clase 131).

Temas

- Encoder + Decoder simétricos.
- Bottleneck: latent space.
- Undercomplete: dimensión chica.
- Stacked: varias capas Dense/Conv.
- Denoising: robusto a ruido.
- Sparse: penalizar ||latent||_1 para que pocas neuronas activas.
- AE para anomaly detection.

Definiciones y características

- Bottleneck: capa con menor dimensión que input, fuerza al modelo a comprimir.
- Reconstruction loss: MSE o BCE pixel-wise.
- Denoising AE: input contaminado → target original. Aprende invariancias.
- Sparse AE: agrega L1 sobre las activaciones latentes a la loss.
- Tied weights: usar W^T del encoder como decoder. Reduce parámetros.

Dataset / recursos

- Fashion-MNIST / MNIST.
- Librerías: tensorflow, keras.

Ejercicios

1. AE simple: Encoder: 784 → 64; Decoder: 64 → 784. Entrenar en MNIST. Visualizar reconstrucciones.
2. Latent space 2D: latent_dim=2. Plot scatter de las representaciones de 1000 imágenes coloreadas por clase.
3. Denoising: noise = 0.5 * rng.normal(x.shape), target = x. Mostrar que reconstruye limpio aunque input está ruidoso.
4. Sparse: agregar keras.regularizers.l1(1e-3) sobre la capa latente. Inspeccionar activaciones.
5. Anomaly detection: entrenar AE solo sobre clase "normal"; calcular reconstruction error en clase "anomalía"; usar como score.

Homework verificable

AE como anomaly detector en Fashion-MNIST:

1. Entrenar AE convolucional solo sobre clase 0 (T-shirt).
2. Calcular MSE de reconstrucción para todas las imágenes test.
3. Plotear histograma del MSE separado por "T-shirt" vs "no T-shirt".
4. ROC-AUC de "es T-shirt" usando MSE como score (negativo).

Criterio de aceptación: ROC-AUC ≥ 0.85; el histograma muestra clara separación.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
latent_dim muy grande → AE copia el input	No comprime nada. Fix: latent_dim < input_
MSE pierde detalle	Lo borronea. Fix: BCE pixel-wise o percept
Decoder con Dense para imágenes → mucho pa	Fix: Conv2DTranspose para upsampling.

Visualizar latent de latent_dim=64 directa	No se puede visualizar 64D. Fix: t-SNE / U
AE para generar nuevas imágenes → outputs	Latent space no es continuo. Fix: VAE (131

Preguntas frecuentes

¿AE moderna?

Sí, especialmente como encoder backbone para auto-supervised pretraining (MAE — Masked Autoencoders en visión, He et al. 2022).

¿AE vs PCA?

Linear AE con MSE = PCA. AE con activaciones no lineales = PCA no lineal. Buena ganancia con datos no lineales.

¿Tied weights?

Reducen params 2× sin perder mucho. Usado históricamente, hoy raro porque GPUs y datos son abundantes.

¿Denoising AE en producción?

Sí, para imagen denoising, audio. Pero modelos de difusión (133) lo hacen mejor.

¿Sparse AE relevante?

Más histórico. Hoy se usan Vector Quantized AE (VQ-VAE) para representaciones discretas (audio, video → tokens).

Referencias

- Géron, cap. 17 — Autoencoders.
- Vincent et al. (2008), Extracting and Composing Robust Features with Denoising Autoencoders.
- He et al. (2022), Masked Autoencoders Are Scalable Vision Learners (MAE), CVPR.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 157 — Clase 157 — Variational Autoencoders (VAE)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 17 § Variational Autoencoders + Kingma & Welling (2014). Duración estimada: 80 min.*

Objetivo

Construir un VAE (Variational Autoencoder, Kingma & Welling 2014) — variante probabilística del AE que aprende una distribución sobre el latent en lugar de un punto: encoder outputs μ, σ de una gaussiana; sampling + reparametrization trick para mantener gradientes. Resultado: latent space continuo y estructurado → permite generación, interpolación entre samples.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Implementar encoder que devuelve (μ, log σ²); sample con z = μ + σ · ε (reparametrization).
- Loss = reconstruction_loss + β · KL(N(μ,σ²) || N(0,I)).
- Generar samples nuevos: muestrear z ~ N(0,I), pasar por el decoder.
- Interpolar en el latent space y verificar transiciones suaves.
- Reconocer que VAE produce outputs borrosos (consecuencia del MSE/BCE) — motivó GANs (132).

Temas

- ELBO (Evidence Lower BOund): log p(x) ≥ E[log p(x|z)] - KL(q(z|x) || p(z)).
- Reparametrization trick: para back-propagar a través del sample.
- β-VAE: subir β fuerza latent más disentangled.
- Posterior collapse: cuando el decoder ignora z.

Definiciones y características

- q(z|x): encoder, devuelve μ(x), σ(x).
- p(z): prior, típicamente N(0, I).
- p(x|z): decoder.
- ELBO: lower bound de log-likelihood que se maximiza.
- KL divergence: KL(N(μ,σ²) || N(0,I)) = 0.5 · Σ (1 + log σ² - μ² - σ²) con signo.
- β-VAE: scale del término KL, controla trade-off reconstrucción vs estructura latente.

Dataset / recursos

- Fashion-MNIST / MNIST / Celeb-A (cara).
- Librerías: tensorflow, keras.

Ejercicios

1. VAE básico: encoder → (z_mean, z_log_var) → sample → decoder. Loss combinada. Entrenar en MNIST.
2. Sampling: muestrear z ~ N(0,I) de tamaño (100, latent_dim). Pasar por decoder. Visualizar las 100 imágenes generadas.
3. Interpolación: dos imágenes A y B → z_A, z_B. Generar 10 imágenes en interpolación lineal entre z_A y z_B. Visualizar.
4. β-VAE: probar β=1, β=5, β=10. Comparar disentanglement vs blurriness.
5. Posterior collapse: con LR alto, el encoder colapsa a μ=0, σ=1. Diagnosticar mirando z_mean.std() cerca de 0.

Homework verificable

VAE sobre Fashion-MNIST:

1. Encoder convolucional, latent_dim=10.
2. Decoder simétrico.
3. Loss: BCE + KL.
4. Entrenar 30 épocas; generar 64 muestras nuevas y visualizar grid.
5. Interpolación entre 2 prendas distintas.

Criterio de aceptación: muestras generadas son reconocibles como prendas (aunque borrosas); interpolación es suave.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Posterior collapse: encoder predice μ=0, σ	KL domina. Fix: KL annealing (subir β grad
Outputs borrosos	Consecuencia inherente de VAE + MSE/BCE. F
KL = 0 → AE puro	β=0 → no es VAE. Fix: β > 0.
Sampleo con z ~ N(μ, σ) en lugar de N(0,I)	Eso es reconstrucción + ruido, no generaci
Generación muestra solo 1 modo	Posterior collapse parcial. Fix: arquitect

Preguntas frecuentes

¿VAE en 2026?

Como generador end-to-end: superado por difusión. Como encoder dentro de Stable Diffusion (latent space comprimido), sí.

¿Por qué reparametrization trick?

Sin él, no se puede back-propagar a través del sample (operación estocástica). Con z = μ + σ·ε, ε es noise externo independiente, gradientes fluyen por μ y σ.

¿Qué es disentanglement?

Cada dimensión latente captura un factor independiente (rotación, color, tamaño). β-VAE y FactorVAE lo persiguen explícitamente.

¿VAE para texto?

Sí, VAE de texto histórico. Hoy LLMs autoregresivos lo cubren mejor.

¿VQ-VAE?

VAE con codebook discreto. Base de muchos modelos modernos: DALL-E 1, Jukebox, EnCodec (audio). Importante.

Referencias

- Géron, cap. 17 — Variational Autoencoders.
- Kingma & Welling (2014), Auto-Encoding Variational Bayes, ICLR.
- Higgins et al. (2017), β-VAE, ICLR.
- van den Oord et al. (2017), Neural Discrete Representation Learning (VQ-VAE), NeurIPS.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 158 — Clase 158 — GANs: DCGAN, Progressive GAN, StyleGAN

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 17 § Generative Adversarial Networks + Goodfellow (2014). Duración estimada: 80 min.*

Objetivo

Construir un GAN (Generative Adversarial Network, Goodfellow 2014) — dos redes en juego adversarial:

Generador crea muestras desde noise; Discriminador distingue real de falso. El equilibrio Nash de este juego = generador que genera muestras indistinguibles de la distribución real. Conocer las variantes principales: DCGAN, Progressive GAN, StyleGAN (caras hiperrealistas).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Implementar un DCGAN sencillo con Generator + Discriminator convolucionales sobre MNIST/Fashion.
- Escribir custom training loop alternando G y D updates.
- Diagnosticar 3 problemas clásicos: mode collapse (G produce 1 sola cosa), vanishing G gradient (D demasiado bueno), training inestable.
- Aplicar label smoothing (real labels = 0.9 en lugar de 1.0), noise en D inputs, y WGAN-GP (Wasserstein) para mejor estabilidad.
- Reconocer que GANs perdieron terreno frente a Diffusion (clase 133) en 2022+, pero StyleGAN sigue siendo competitivo para caras.

Temas

- Loss original (min-max): min_G max_D E[log D(x)] + E[log(1 - D(G(z)))].
- DCGAN: arquitecturas convolucionales (Radford et al. 2015).
- Modos de fallo: mode collapse, D demasiado fuerte/débil.
- WGAN, WGAN-GP, spectral norm.
- Progressive GAN: empezar con 4×4, ir subiendo resolución.
- StyleGAN: AdaIN, style mixing, latent W intermedio.
- Métricas: FID (Fréchet Inception Distance), IS.

Definiciones y características

- Generator: red que toma z ~ N(0,I) → genera muestra.
- Discriminator: clasificador binario real/fake.
- Mode collapse: G genera diversidad reducida (1 o pocos modos).
- Vanishing gradient (G): cuando D distingue perfectamente, su gradiente sobre G se desvanece.
- WGAN-GP: cambia la loss a Wasserstein-1 + gradient penalty. Estabilidad superior.
- FID: distancia entre features de InceptionV3 de reales vs falsas; menor = mejor.
- StyleGAN: latent z se mapea a w y se inyecta vía AdaIN en cada capa del generador.

Dataset / recursos

- MNIST / Fashion-MNIST como playground.
- Celeb-A para caras (más serio).
- Librerías: tensorflow, keras.

Ejercicios

1. DCGAN básico: G y D convolucionales. Custom training loop con dos tf.function (train_d, train_g).
2. Diagnóstico: graficar D_loss y G_loss por step. Si D_loss → 0, G está perdiendo.
3. Mode collapse: tras N épocas, generar 64 samples. Si todos son la misma cosa → collapse.
4. WGAN-GP: implementar gradient penalty en la D loss. Comparar estabilidad vs DCGAN.
5. FID: implementar (o usar tensorflow_gan.eval.fid) y reportar FID del modelo.

Homework verificable

DCGAN en Fashion-MNIST:

1. Generator: Dense(77128) → reshape → Conv2DTranspose × 3 hasta 28×28.

2. Discriminator: Conv2D × 3 → Dense(1).
3. Entrenar 50 épocas; generar grid 8×8.
4. Reportar D_loss y G_loss finales; visual del grid.

Criterio de aceptación: las muestras generadas son reconocibles como prendas (aunque imperfectas); el training es razonablemente estable (sin colapsar a un solo modo).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Mode collapse	D muy fuerte, G no puede engañar. Fix: WGA
Training inestable, oscilaciones	LR alto en ambos. Fix: Adam(2e-4, beta_1=0
D_loss = 0	D resolvió. Fix: hacer D más débil o agreg
Outputs muy borrosos	Underfit de G. Fix: G más expresivo, más é
Outputs de baja resolución bien, alta reso	Sin Progressive Growing. Fix: usar StyleGA

Preguntas frecuentes

¿GAN sigue relevante en 2026?

StyleGAN3 sigue siendo competitivo para caras y generación en general "rápida" (1 forward pass). Difusión gana en calidad para texto-to-image complejo.

¿FID o IS?

FID es mejor (más robusta a artifacts). IS está deprecada.

¿Conditional GAN?

cGAN (Mirza & Osindero 2014): condicionar en label o texto. Permite control sobre qué generar.

¿GAN inversion?

Encontrar z tal que G(z) ≈ imagen_dada. Útil para edición. Difícil; StyleGAN tiene buenas implementaciones.

¿Por qué Diffusion ganó?

Más estable de entrenar (sin min-max), mejor calidad de outputs, controllable con text. La explicación más simple: optimiza log p(x) directamente, GAN optimiza un divergence relacionado pero no la log-likelihood.

Referencias

- Géron, cap. 17 — Generative Adversarial Networks.
- Goodfellow et al. (2014), Generative Adversarial Networks, NeurIPS.
- Radford et al. (2015), DCGAN, ICLR.
- Karras et al. (2018), Progressive Growing of GANs, ICLR.
- Karras et al. (2019, 2020, 2021), StyleGAN1/2/3, NeurIPS/CVPR.
- Gulrajani et al. (2017), WGAN-GP, NeurIPS.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 159 — Clase 159 — Modelos de difusión (+ Stable Diffusion XL, ControlNet, LCM)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 17 § Diffusion Models + Ho et al. (2020) DDPM + papers SDXL, ControlNet, LCM. Duración estimada: 105 min.*

Objetivo

Entender modelos de difusión — la familia que destronó a GANs en 2022 (DALL-E 2, Stable Diffusion, Midjourney). Idea: forward process agrega ruido gaussiano gradualmente; reverse process (aprendido) lo elimina paso a paso. Conocer DDPM clásico, latent diffusion (Stable Diffusion), ControlNet para condicionamiento espacial, LCM (Latent Consistency Models) para inference rápida.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Implementar un DDPM sencillo: forward q(x_t | x_0), U-Net que predice el ruido ε_θ(x_t, t).
- Aplicar el sampling DDPM: 1000 steps de denoising desde x_T ~ N(0,I).
- Reconocer latent diffusion: aplicar difusión en el espacio comprimido de un VAE (1/64 del tamaño) → 8× más rápido.
- Usar Stable Diffusion XL con diffusers library: prompt → imagen en 3 líneas.
- Aplicar ControlNet para condicionar generación en pose, edge, depth, segmentation.
- Acelerar inference con LCM o Turbo (1-4 steps en lugar de 50).

Temas

- Forward process: q(x_t | x_{t-1}) = N(√(1-β_t) x_{t-1}, β_t I).
- Closed form: q(x_t | x_0) = N(√ᾱ_t · x_0, (1-ᾱ_t) I).
- Loss simple (Ho 2020): MSE(ε, ε_θ(x_t, t)) — predecir el ruido.
- U-Net architecture (encoder-decoder con skip connections).
- Sampling DDPM vs DDIM vs DPM-Solver: 1000 → 50 → 20 steps.
- Complemento moderno: Stable Diffusion XL, ControlNet, LCM/Turbo.

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 133a — Stable Diffusion XL + ControlNet en profundidad

Definiciones y características

- Forward (diffusion) process: q(x_t | x_{t-1}) agrega ruido. Markov chain.
- Reverse process: p_θ(x_{t-1} | x_t) aprendido, denoise.
- Schedule: β_t o ᾱ_t — cuánto ruido en cada step. Linear o cosine.
- U-Net: arquitectura encoder-decoder con skip connections, usada como denoiser.
- DDIM: sampler deterministic más rápido que DDPM (Song et al. 2021).
- Classifier-Free Guidance (CFG): en inference, mezcla condicional + uncondicional para control.
- ControlNet: adapter para control espacial.
- LCM: distillation a 1-4 steps.

Dataset / recursos

- Fashion-MNIST como playground.

- HF Hub para modelos pre-trained.
- Librerías: diffusers, transformers, accelerate, controlnet_aux.

Ejercicios

1. DDPM en MNIST: U-Net chico + DDPM scheduler. Entrenar 50 épocas; samplear 64 imágenes.
2. SDXL inference: cargar SDXL y generar 4 imágenes desde prompts.
3. CFG: variar guidance_scale {1, 5, 15}. Ver cómo afecta fidelidad al prompt vs diversidad.
4. ControlNet Canny: tomar una foto, extraer bordes con OpenCV, generar variantes condicionadas con SDXL+ControlNet.
5. LCM: comparar SDXL base (30 steps) vs SDXL + LCM-LoRA (4 steps). Tiempo y calidad.

Homework verificable

Pipeline de generación SDXL controlado:

1. Imagen base (e.g., una foto de un edificio).
2. Extraer Canny con OpenCV.
3. Generar 3 variantes con SDXL+ControlNet con prompts distintos.
4. Comparar tiempo SDXL vs SDXL+LCM.

Criterio de aceptación: las 3 variantes preservan la estructura del edificio según los Canny; LCM es ≥ 5× más rápido con calidad razonable.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
OOM al cargar SDXL en GPU 8 GB	Modelo muy grande. Fix: fp16, enable_atten
Imagenes borrosas con LCM	Steps insuficientes. Fix: subir a 4-8; aju
ControlNet no obedece el control	controlnet_conditioning_scale muy bajo. Fi
Generación con prompts pobres da resultado	Prompt engineering es real. Fix: estilo "a
Difusión propia entrenada genera ruido	No converge. Fix: verificar la schedule, U

Preguntas frecuentes

¿GAN o Diffusion en 2026?

Diffusion gana en calidad y controlabilidad. GAN (StyleGAN3) sigue siendo competitivo para caras (1 forward pass).

¿DDPM o DDIM o DPM-Solver?

Para inference: DPM-Solver / DPM++ (Lu et al. 2022) — 20 steps con misma calidad que DDPM 1000. Default en diffusers.

¿Cómo fine-tunear SDXL en estilo propio?

LoRA sobre el U-Net con diffusers examples. 20-50 imágenes alcanza para un estilo. ~1 hora en GPU consumer.

¿Difusión para video?

Sí. Sora, Veo, Stable Video Diffusion. Más caro computacionalmente.

¿Difusión para texto?

Sí pero menos competitivo que LLMs autoregresivos. Áreas activas de investigación.

Referencias

- Géron, cap. 17 — Diffusion Models.
- Ho, Jain & Abbeel (2020), Denoising Diffusion Probabilistic Models, NeurIPS.
- Rombach et al. (2022), Stable Diffusion, CVPR.
- Zhang, Rao & Agrawala (2023), ControlNet, ICCV.
- Luo et al. (2023), Latent Consistency Models.
- diffusers docs.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 160 — Clase 160 — Stable Diffusion XL + ControlNet en profundidad

> *Parte: 2 — Deep Learning · Fuente: Rombach et al. (2022) SD + Podell et al. (2023) SDXL + Zhang et al. (2023) ControlNet. Duración estimada: 90 min.*

Objetivo

Dominar Stable Diffusion XL (Stability AI 2023) en producción: pipeline completo (text encoder dual, U-Net, VAE), schedulers modernos (DPM-Solver++, Euler ancestral, UniPC), CFG (Classifier-Free Guidance), prompt weighting. Combinar con ControlNet para condicionamiento espacial (Canny, depth, pose, segmentation, lineart). Conocer Flux (Black Forest Labs 2024) como sucesor open-source.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Cargar SDXL: StableDiffusionXLPipeline.from_pretrained('stabilityai/stable-diffusion-xl-base-1.0').
- Aplicar refiner opcional para detalle final.
- Usar schedulers distintos y entender trade-off speed/quality.
- Agregar ControlNet (canny, depth, openpose) sobre SDXL.
- Aplicar LoRA para estilo custom (pipe.load_lora_weights('path')).
- Reconocer Flux y SD3 como sucesores.

Temas

- Pipeline SDXL: text_encoder_1 (CLIP-L), text_encoder_2 (CLIP-G), U-Net 2.6B params, VAE.
- Schedulers: DDIM (50 steps), DPM-Solver++ (20 steps), Euler ancestral (creative), UniPC (10-20 steps).
- CFG: guidance_scale=7.5 default; > 12 over-fitting al prompt.
- Negative prompts.
- Refiner (paso opcional adicional).
- ControlNet variantes: Canny, Depth, OpenPose, Scribble, Lineart, MLSD, Tile.

Definiciones y características

- SDXL: 2.6B params U-Net, latent space 128×128×4 desde imagen 1024×1024.
- Dual text encoder: CLIP-L (768d) + CLIP-G (1280d) concatenados.
- CFG: output = uncond + scale · (cond - uncond).

- Negative prompt: lo que NO querés (ugly, blurry, low quality).
- ControlNet: red adicional, freezeada o trained, que inyecta condicionamiento espacial.
- LoRA: rank-low adapter, ~50-200 MB, para estilos.
- Flux 1: sucesor open-source SDXL (2024), mejor calidad y prompt following.

Dataset / recursos

- HuggingFace: stabilityai/stable-diffusion-xl-base-1.0, ControlNets en diffusers/controlnet-*-sdxl-1.0.
- Librerías: diffusers, transformers, accelerate, controlnet_aux.

Ejercicios

1. SDXL básico: prompt → imagen 1024². Comparar schedulers (DPM-Solver++, Euler a, UniPC).
2. CFG sweep: guidance_scale {3, 7.5, 12, 18}. Ver trade-off creativity/fidelity.
3. Negative prompt: agregar "blurry, watermark, low quality" y comparar.
4. ControlNet Canny: extraer Canny de una foto, generar variante manteniendo estructura.
5. LoRA estilo: cargar un LoRA de estilo (CivitAI), aplicar. Variar cross_attention_kwargs={'scale': 0.8}.

Homework verificable

Pipeline visual: SDXL + ControlNet Canny + LoRA estilo:

1. Foto base → Canny.
2. SDXL + ControlNet, prompt + LoRA estilo.
3. Generar 4 variantes; comparar.
4. Reportar tiempo y memoria.

Criterio de aceptación: estructura preservada por Canny; estilo del LoRA visible; outputs de buena calidad.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
OOM en GPU 8 GB	Modelo grande. Fix: pipe.enable_attention_
Outputs raros, deformados	CFG demasiado alto. Fix: guidance_scale=5-
ControlNet no obedece estructura	controlnet_conditioning_scale muy bajo. Fi
Manos deformadas	Bug clásico de difusión. Fix: LoRA específ
LoRA no se siente	Scale muy bajo. Fix: cross_attention_kwarg

Preguntas frecuentes

SDXL o Flux?

Flux (2024) mejor calidad y prompt following. SDXL más maduro, más LoRAs/ControlNets disponibles. Ambos viables.

SD3 dónde está?

Stability AI lo publicó pero con licencia restrictiva → comunidad migró a Flux. SD3.5 medium open license.

Refiner vale la pena?

A veces. Toma más tiempo. Probar con vs sin para tu caso.

Cómo entreno LoRA para SDXL?

diffusers examples + dataset de 10-30 imágenes + script de training. 1-2 horas GPU.

Inpainting?

StableDiffusionXLInpaintPipeline. Modificar regiones específicas con máscara.

Referencias

- Podell et al. (2023), SDXL.
- Zhang et al. (2023), ControlNet, ICCV.
- diffusers docs.
- Flux (Black Forest Labs).
- CivitAI — comunidad LoRA/checkpoints.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 161 — Clase 161 — RL: aprendizaje por recompensa, Gymnasium (Farama)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 18 § Introduction to Reinforcement Learning + docs Gymnasium. Duración estimada: 75 min.*

Objetivo

Entender el paradigma de Reinforcement Learning — un agente interactúa con un environment, observa states, toma actions, recibe rewards, y aprende una policy que maximiza recompensa acumulada. Conocer Gymnasium (Farama Foundation, fork del antiguo OpenAI Gym que dejó de mantenerse en 2022) — la librería estándar de environments para benchmarks.

> *Nota: el nombre "OpenAI Gym" es histórico. Desde 2022, OpenAI dejó de mantenerlo y la Farama Foundation tomó el relevo creando Gymnasium — drop-in replacement con mejor mantenimiento.*

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Definir los 5 componentes RL: agent, environment, state, action, reward.
- Usar gymnasium: env = gym.make('CartPole-v1'); obs, _ = env.reset(); obs, reward, terminated, truncated, info = env.step(action).
- Implementar un policy random como baseline.
- Reconocer la diferencia entre on-policy (PPO, A2C) y off-policy (DQN, SAC).
- Saber dónde RL es apropiado (juegos, robótica, navegación) vs donde no (clasificación, regresión típicas).

Temas

- Bloque básico: state → action → reward → next_state.
- Episodic vs continuous tasks.
- Discrete vs continuous action spaces.
- Discount factor γ: balance entre recompensa inmediata y futura.
- Gymnasium API estándar (reset, step).
- Environments populares: CartPole, MountainCar, Atari, MuJoCo, BipedalWalker.

Definiciones y características

- Policy π(a|s): la estrategia — distribución sobre acciones dado state.
- Trajectory / episode: secuencia (s_0, a_0, r_0, s_1, a_1, ...) hasta done.
- Return: suma de recompensas futuras G_t = Σ γ^k r_{t+k}.
- Value function V(s): expected return desde s siguiendo π.
- Q-function Q(s, a): expected return tras tomar a desde s.
- Gymnasium step: devuelve 5-tuple (obs, reward, terminated, truncated, info) (Gym original devolvía 4 — done combinaba terminated/truncated).
- Action space: Discrete(N) o Box(low, high, shape).

Dataset / recursos

- gymnasium library (pip install gymnasium).
- Environments built-in: CartPole-v1, LunarLander-v3, MountainCar-v0.
- Librerías: gymnasium, numpy, matplotlib.

Ejercicios

1. Entorno básico: env = gym.make('CartPole-v1', render_mode='human'). Correr 10 episodios con acciones random; reportar duración promedio.
2. Estructura del state: imprimir env.observation_space y env.action_space para CartPole, MountainCar, LunarLander.
3. Policy heurística: para CartPole, action = 0 if pole_angle < 0 else 1 (push opuesto al tilt). Comparar contra random.
4. Return discounted: calcular G_t = Σ γ^k r_{t+k} con γ=0.99 sobre un episodio.
5. Render: usar render_mode='rgb_array' y guardar frames para crear un gif.

Homework verificable

Sobre CartPole-v1:

1. Implementar 3 policies: random, heurística simple, "siempre derecha".
2. Correr 100 episodios de cada; reportar return promedio y desviación.
3. Verificar que heurística supera random.

Criterio de aceptación: random ≈ 20 steps; heurística ≥ 80 steps; "siempre derecha" muere rápido.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
gym.make(...) deprecated warning	Estás usando el viejo OpenAI Gym. Fix: pip
env.step(action) devuelve 4 valores	Gym viejo. Gymnasium devuelve 5. Fix: actu
env.reset() devuelve un solo valor en tuto	Gym pre-2022 devolvía obs. Gymnasium devue
Render no muestra ventana	render_mode no especificado al crear env.
Atari environments fail	Requieren pip install gymnasium[atari,acce

Preguntas frecuentes

¿RL vs supervised?

Supervised: aprende f(x) = y con labels. RL: aprende π(s) = a con feedback indirecto (rewards). Más cercano a humanos pero más difícil.

¿Cuándo RL?

Juegos (AlphaGo), robótica, navegación, sistemas de recomendación con feedback de usuarios. NO para clasificación / regresión típicas.

¿Gymnasium o RLlib o Stable-Baselines?

Gymnasium = environments. Stable-Baselines3 (PyTorch) o RLlib (Ray) = algoritmos. Estándar 2026: SB3 para experimentos, RLlib para producción a escala.

¿RL con LLMs?

Sí — RLHF y DPO (clase 128) son aplicaciones de RL a LLMs.

¿Cuánto cuesta entrenar?

CartPole: 1 minuto. Atari: horas-días. AlphaGo: semanas en cluster. RL es muy costoso en datos (interacciones).

Referencias

- Géron, cap. 18 — Reinforcement Learning.
- Sutton & Barto (2018), Reinforcement Learning: An Introduction (2nd ed.) — biblia del RL.
- Gymnasium docs.
- Stable-Baselines3 docs.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 162 — Clase 162 — Policy gradients

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 18 § Policy Gradients + Sutton & Barto, cap. 13. Duración estimada: 70 min.*

Objetivo

Implementar policy gradient —REINFORCE (Williams 1992)—: parametrizar la policy con una red neuronal π_θ(a|s), optimizar directamente la expected return via gradiente. Es el método más simple de RL que usa redes y la base conceptual de PPO/A2C/A3C (clase 138).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Definir la policy como red state → softmax(actions).
- Calcular el gradiente REINFORCE: θ J = E[θ log π_θ(a|s) · G_t].
- Implementar el training loop: rollout → calcular returns → gradient ascent.
- Aplicar baseline (substraer V(s) de G_t) para reducir varianza.
- Reconocer la limitación: alta varianza, lento (motiva A2C/PPO).

Temas

- Expected return J(θ) = E_π[G].
- Policy gradient theorem: θ J = E[θ log π · Q].

- REINFORCE algorithm: rollout completo + apply gradient.
- Baseline para reducir varianza.
- Discounted returns con γ.

Definiciones y características

- Policy network: π_θ(a|s), típicamente MLP con softmax (discrete actions) o gaussiana (continuas).
- Log-prob: log π(a|s), lo que multiplicamos por G_t para el gradient.
- Discounted return G_t: Σ γ^k r_{t+k}.
- Baseline: cualquier función de s (e.g., V(s)) que reduce varianza sin sesgo.
- Advantage A = G - V(s): cuán mejor o peor fue la acción comparada con el valor esperado.

Dataset / recursos

- CartPole-v1.
- Librerías: gymnasium, tensorflow, keras.

Ejercicios

1. Policy network: Dense(32) → Dense(32) → Dense(2, softmax) para CartPole.
2. Rollout: ejecutar 1 episodio, guardar (s, a, r) por timestep.
3. Returns: calcular G_t para cada timestep con γ=0.99.
4. Gradient step: loss = -Σ log π(a_t|s_t) · G_t; backward; apply.
5. Con baseline: agregar V(s) head, restar de G antes del gradient.

Homework verificable

REINFORCE en CartPole:

1. Policy Dense(64) → Dense(64) → Dense(2, softmax).
2. Train 500 episodios.
3. Reportar return medio por época.
4. Comparar con/sin baseline.

Criterio de aceptación: episode return llega a ≥ 195 (resuelto) en ≤ 300 episodios; baseline acelera convergencia.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Alta varianza en gradientes	Inherente a REINFORCE. Fix: baseline, batc
Policy collapse a una acción	LR alto o sin entropy bonus. Fix: bajar LR
Recompensas muy chicas → updates débiles	Normalizar G (mean=0, std=1) por episode.
Action probs nan	Inputs sin normalizar + softmax. Fix: clip
El env tiene rewards muy variables → entre	Probar con env más simple primero.

Preguntas frecuentes

¿REINFORCE en producción?

Casi no — pero es la base. PPO es el default industrial moderno (clase 138).

¿On-policy o off-policy?

REINFORCE es on-policy (entrena con datos generados por la policy actual). Off-policy (DQN, SAC) reutiliza datos viejos.

¿Cómo elijo γ?

0.99 default. Más alto = más visión a largo plazo, pero más varianza. Para tareas episódicas cortas, 0.95-0.99.

¿Entropy regularization?

Agregar -β · H(π) a la loss promueve exploración (policy no determinística temprano). Estándar.

¿Cómo veo si converge?

Plot del return promedio por epoch (smoothed). Sube → converge.

Referencias

- Géron, cap. 18 — Policy Gradients.
- Williams (1992), Simple Statistical Gradient-Following Algorithms (REINFORCE).
- Sutton & Barto (2018), cap. 13.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 163 — Clase 163 — Markov Decision Processes

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 18 § Markov Decision Processes + Sutton & Barto cap. 3. Duración estimada: 60 min.*

Objetivo

Formalizar el marco teórico de RL: un Markov Decision Process (MDP) = tupla (S, A, P, R, γ). Conocer la Bellman equation que define V y Q óptimos, y los algoritmos clásicos Value Iteration y Policy Iteration que los resuelven (cuando el MDP es conocido).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Definir un MDP: states, actions, transition prob, reward function, discount.
- Escribir la Bellman equation para V: V(s) = max_a Σ P(s'|s,a)[R + γ V*(s')].
- Implementar Value Iteration: actualizar iterativamente hasta convergencia.
- Implementar Policy Iteration: alternar policy evaluation + policy improvement.
- Reconocer la limitación: requiere conocer P y R (no aplicable a entornos reales) → motivó model-free (Q-learning, clase 137).

Temas

- Propiedad de Markov: P(s_{t+1} | s_t, a_t) = P(s_{t+1} | s_t, a_t, s_{t-1}, ...).
- Componentes MDP.
- Bellman optimality equation.
- Value Iteration (synchronous update).
- Policy Iteration (alternar eval + improve).

- Convergencia garantizada (contractive operator).

Definiciones y características

- State s: representación del entorno.
- Action a: lo que el agente puede hacer.
- Transition P(s' | s, a): probabilidad de llegar a s' desde s tomando a.
- Reward R(s, a, s'): recompensa inmediata.
- Discount γ: peso de recompensas futuras.
- V*(s): máximo expected return desde s.
- Q*(s, a): máximo expected return tras tomar a en s.
- Policy óptima: π(s) = argmax_a Q(s, a).

Dataset / recursos

- MDPs sintéticos chicos (FrozenLake, Taxi de Gymnasium).
- Librerías: gymnasium, numpy.

Ejercicios

1. MDP de juguete: definir un MDP de 4 estados con P, R manualmente.
2. Value Iteration: implementar V[s] = max_a Σ P(s'|s,a)(R + γ V[s']) hasta max change < 1e-6.
3. Policy Iteration: alternar evaluación (V^π) con mejora (π' = greedy(V)) hasta estabilidad.
4. FrozenLake: cargar gym.make('FrozenLake-v1'), extraer env.unwrapped.P (modelo del MDP), resolver con VI.
5. Compare: # iteraciones VI vs PI para llegar a misma policy.

Homework verificable

Sobre FrozenLake-v1 (is_slippery=True):

1. Extraer modelo P y R desde env.unwrapped.P.
2. Implementar Value Iteration; reportar V y π greedy.
3. Evaluar la policy con 1000 episodios random; reportar success rate.

Criterio de aceptación: success rate ≥ 0.7 sobre FrozenLake slippery; V* y policy claramente prefieren caminos seguros.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Value Iteration no converge	γ ≥ 1.0. Fix: γ < 1.0 (típicamente 0.95-0.
Policy Iteration loops infinito	Numerical issues en evaluación. Fix: limit
Acceder a env.P falla	Hay que usar env.unwrapped.P.
Asumir Markov en problemas non-Markov	Por ejemplo, control con velocidad escondi
VI vs PI: cuándo usar?	VI: más simple, más iters pero más baratas

Preguntas frecuentes

¿MDPs en problemas reales?

Casi nunca conocés P y R exactamente. Por eso → Q-learning (clase 137) y métodos model-free.

¿POMDPs?

Partially Observable: cuando no ves el state completo. Mucho más difícil. Usar agente con memoria (LSTM,

Transformer).

¿Continuous state space?

VI/PI no escalan. Usar function approximation: DQN (clase 137) o policy gradient (135).

¿Bellman equation conexión con DP?

Sí, VI es dynamic programming clásico aplicado a MDPs.

¿Aprender el modelo P, R desde data?

Model-based RL: aprender un modelo, planificar con él. Más sample-efficient pero más complejo.

Referencias

- Géron, cap. 18 — Markov Decision Processes.
- Sutton & Barto, cap. 3-4.
- Bellman (1957), Dynamic Programming.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 164 — Clase 164 — TD Learning, Q-Learning, Deep Q-Networks

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 18 § Q-Learning y § Deep Q-Learning. Duración estimada: 80 min.*

Objetivo

Implementar Q-Learning clásico (Watkins 1989) y su versión moderna DQN (Mnih et al. 2015, Nature paper que aprendió Atari desde pixels). Off-policy, model-free, bootstrap. Conocer los 2 trucos que hicieron a DQN funcionar: experience replay y target network.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar la update Q-learning: Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') - Q(s,a)].
- Implementar Q-learning tabular en FrozenLake.
- Construir un DQN: red state → Q(a) para todas las actions, MSE entre Q predicted y r + γ max_a' Q'(s',a').
- Implementar replay buffer (almacenar transitions, samplear batch para training).
- Implementar target network (copia frozen actualizada cada N steps).

Temas

- TD (Temporal Difference) error: δ = r + γ V(s') - V(s).
- Q-learning: off-policy, sigue greedy de Q*.
- ε-greedy exploration: con prob ε explora random, sino greedy.
- Replay buffer: deque de (s, a, r, s', done).
- Target network: estabilidad (sino, target se mueve mientras estimás).

- DQN sobre CartPole y Atari.

Definiciones y características

- Q-value Q(s, a): expected return tras tomar a desde s.
- Bootstrap: update usando estimación actual (sin esperar fin de episodio).
- Off-policy: aprende sobre policy óptima aunque exploración sea ε-greedy.
- Replay buffer: almacena experiencias para sampleo iid.
- Target network Q': copia de Q usada para calcular el target. Actualizada cada N steps.
- ε-greedy: balance exploration vs exploitation.

Dataset / recursos

- FrozenLake (tabular).
- CartPole (DQN).
- Librerías: gymnasium, tensorflow, keras, numpy.

Ejercicios

1. Q-learning tabular: en FrozenLake, mantener Q[s, a] numpy array. Update con ε-greedy. Reportar success rate tras 5000 episodios.
2. DQN básico: red Dense(64) → Dense(64) → Dense(2) para CartPole. Sin replay buffer ni target network → ver inestabilidad.
3. Replay buffer: from collections import deque; buffer = deque(maxlen=10_000). Sample batch=32 random.
4. Target network: copiar Q.weights cada 100 steps a Q_target.
5. ε decay: empezar ε=1.0, decaer linealmente a 0.01 en 10 000 steps.

Homework verificable

DQN sobre CartPole-v1:

1. Red Dense(64, relu) → Dense(64, relu) → Dense(2).
2. Replay buffer 50 000, batch 64.
3. Target network sync cada 100 steps.
4. ε: linear decay 1.0 → 0.05 sobre 10 000 steps.
5. Train hasta mean_reward(100 episodios) ≥ 195.

Criterio de aceptación: alcanza ≥ 195 en ≤ 500 episodios; gráfico de return promedio muestra convergencia.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Q values explotan	Sin target network, bootstrap inestable. F
Convergencia lenta	Replay buffer muy chico o batch muy chico.
No explora → policy subóptima	ε muy chico desde el inicio. Fix: ε=1.0 al
done confundido con truncated	Gymnasium 5-tuple. Fix: tratar terminated,
OOM con Atari	Stacks de 4 frames + buffer enorme. Fix: r

Preguntas frecuentes

¿DQN supera Q-learning tabular cuándo?

Cuando state space es enorme (continuous o pixels). Para FrozenLake (16 states), tabular gana.

¿Variantes de DQN?

- Double DQN (van Hasselt 2016): reduce overestimación.
- Dueling DQN (Wang 2016): separa V y advantage.
- Prioritized Experience Replay: sampleo no uniforme.
- Rainbow (Hessel 2018): combina todas.

¿DQN vs Policy Gradient?

DQN: off-policy, sample-efficient, action space discreto. PG (PPO): on-policy, más estable, action space continuo. Ambos en el zoo moderno.

¿Por qué replay buffer rompe la correlación temporal?

Sin él, transitions consecutivas están correlacionadas → batches no iid → gradiente sesgado.

¿Cuál env empezar?

CartPole para entender. LunarLander para algo más complejo. Atari para serio (requiere días).

Referencias

- Géron, cap. 18 — Q-Learning y Deep Q-Learning.
- Watkins (1989), Learning from Delayed Rewards (PhD thesis).
- Mnih et al. (2015), Human-level control through deep reinforcement learning, Nature — DQN paper.
- van Hasselt et al. (2016), Deep Reinforcement Learning with Double Q-learning.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 165 — Clase 165 — RL moderno: A3C, PPO, SAC (vista general)

> *Parte: 2 — Deep Learning · Fuente: papers A3C (Mnih 2016), PPO (Schulman 2017), SAC (Haarnoja 2018) + docs Stable-Baselines3. Duración estimada: 65 min.*

Objetivo

Vista general (sin implementación desde cero) de los 3 algoritmos modernos de RL: A3C (Asynchronous Advantage Actor-Critic), PPO (Proximal Policy Optimization — el default industrial), y SAC (Soft Actor-Critic — off-policy, continuous actions). Saber cuál elegir y cómo usarlos con Stable-Baselines3.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diferenciar on-policy (A3C, PPO) de off-policy (SAC, DQN).
- Reconocer la idea de Actor-Critic: dos redes — actor (policy) + critic (value).
- Entender el clipped objective de PPO: limita updates a [1-ε, 1+ε] veces el old policy → estabilidad.
- Usar Stable-Baselines3: PPO('MlpPolicy', env).learn(total_timesteps=100_000).
- Elegir: PPO para discreto/continuo, on-policy. SAC para continuo, off-policy, sample-efficient.

Temas

- Actor-Critic: actor da policy, critic da V(s). Advantage = G - V(s).
- A3C: async + advantage actor-critic. Paralelización con multiples workers.
- A2C: variante sincrónica (más simple).
- PPO: clipped surrogate objective.
- SAC: actor-critic off-policy + entropy regularization.
- Stable-Baselines3 como librería estándar.

Definiciones y características

- Actor: red de policy π_θ(a|s).
- Critic: red de valor V_φ(s) o Q_φ(s,a).
- Advantage A(s, a): cuán mejor es a que el promedio.
- GAE (Generalized Advantage Estimation): estimador robusto del advantage usado en PPO.
- PPO clip: min(ratio · A, clip(ratio, 1-ε, 1+ε) · A) con ratio = π_new/π_old.
- SAC entropy bonus: H(π) se maximiza junto al return → exploración natural.

Dataset / recursos

- LunarLander, CartPole, Pendulum.
- Librerías: gymnasium, stable-baselines3 (pip install stable-baselines3[extra]).

Ejercicios

1. PPO con SB3: from stable_baselines3 import PPO; model = PPO('MlpPolicy', 'CartPole-v1', verbose=1); model.learn(50_000). Evaluar.
2. SAC en Pendulum: SAC('MlpPolicy', 'Pendulum-v1').learn(20_000).
3. Comparar: PPO vs SAC en LunarLander; reportar return y sample efficiency.
4. TensorBoard: PPO(..., tensorboard_log='./tb/') y ver curvas.
5. Custom callback: EvalCallback para evaluar cada N steps y guardar mejor modelo.

Homework verificable

Entrenar PPO en LunarLander-v3:

1. PPO('MlpPolicy', 'LunarLander-v3', verbose=1, tensorboard_log='./tb/').
2. model.learn(total_timesteps=500_000).
3. Evaluar 100 episodios; reportar mean reward.

Criterio de aceptación: mean reward ≥ 200 (problema resuelto); TensorBoard muestra curva ascendente.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
PPO no converge	LR alto o env muy complejo. Fix: learning_
SAC en discrete actions	SAC es continuous. Fix: PPO o DQN para dis
Reward "stuck" en ~0	El env tal vez no es resoluble con MLP. Fi
Training muy lento	Pocos parallel envs. Fix: make_vec_env('Ca
Modelo guardado no carga	Versión de SB3 incompatible. Fix: model =

Preguntas frecuentes

¿PPO o SAC?

- PPO: discreto o continuo, on-policy, más estable, default industrial.
- SAC: continuo, off-policy, sample-efficient, mejor performance pico.

- Estándar: PPO primero; SAC si necesitás más sample efficiency.

¿Cuántos steps?

CartPole: 50k. LunarLander: 500k. Atari: 10M+. Robótica MuJoCo: 1M-10M.

¿RL para LLM (RLHF)?

PPO es el algo estándar de RLHF. DPO (clase 128) lo reemplazó por simplicidad.

¿Cuándo NO usar RL?

Si el problema es supervised (clasificación, regresión), no uses RL. RL es sample-inefficient y caro.

¿Hyperparameter tuning?

Optuna sobre SB3 o rl_zoo3 que tiene hyperparams pre-tuneados para muchos envs.

Referencias

- Mnih et al. (2016), Asynchronous Methods for Deep Reinforcement Learning (A3C), ICML.
- Schulman et al. (2017), PPO.
- Haarnoja et al. (2018), Soft Actor-Critic, ICML.
- Stable-Baselines3 docs.
- Sutton & Barto, capítulos avanzados.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 166 — Clase 166 — TF Serving + gRPC (+ ONNX, TensorRT, vLLM/TGI)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 19 § Deploying a Model to a Service + docs ONNX, TensorRT, vLLM, TGI. Duración estimada: 100 min.*

Objetivo

Servir un modelo entrenado a producción. Aprender TF Serving (oficial de TensorFlow, gRPC/REST, batching) y las alternativas modernas multi-framework: ONNX + ONNX Runtime (portable a cualquier framework / runtime), TensorRT (NVIDIA, máxima velocidad en GPU NVIDIA), y para LLMs: vLLM y TGI (Text Generation Inference) — específicos del caso autoregresivo con continuous batching.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Exportar un modelo Keras a SavedModel: model.save('servable/1/', save_format='tf').
- Levantar TF Serving con Docker: docker run -p 8501:8501 -v $PWD/servable:/models/m tensorflow/serving --model_name=m.
- Hacer requests REST/gRPC.
- Exportar a ONNX con tf2onnx / torch.onnx.export, servir con ONNX Runtime.
- Conocer cuándo usar TensorRT (latencia mínima en GPU NVIDIA) o vLLM/TGI (LLMs).

Temas

- SavedModel format (TF nativo).
- TF Serving: configuración, versioning, model warm-up, batching.
- gRPC vs REST: gRPC más rápido (binary protobuf); REST más simple.
- Complemento moderno: ONNX/ONNX Runtime, TensorRT, vLLM/TGI.

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 139a — ONNX y ONNX Runtime: portabilidad e inference optimizada

Definiciones y características

- SavedModel: formato TF para modelos servibles (variables + graph + signature).
- Signature: contrato del modelo (predict con inputs/outputs nombrados).
- TF Serving: server escrito en C++, batching automático, versioning.
- gRPC: protocolo binario sobre HTTP/2. Más rápido que REST.
- ONNX: formato intermedio standard.
- TensorRT: optimizador NVIDIA específico.
- Triton: servidor multi-framework.

Dataset / recursos

- Un modelo ya entrenado de las clases previas (Fashion-MNIST MLP).
- Librerías: tensorflow, tensorflow-serving-api, tf2onnx, onnxruntime, opcional tensorrt.

Ejercicios

1. Export SavedModel: model.export('servable/1/') (Keras 3). Inspeccionar la estructura assets, variables, saved_model.pb.
2. TF Serving con Docker: levantar el container con el modelo montado, hacer una request REST a localhost:8501/v1/models/m:predict.
3. ONNX export: convertir el TF model a ONNX con tf2onnx. Cargar con ONNX Runtime y verificar misma predicción.
4. vLLM: levantar vllm con mistralai/Mistral-7B-Instruct. Cliente OpenAI-style. Medir tokens/sec.
5. Latencia comparada: TF Serving REST vs gRPC vs ONNX Runtime CPU vs TensorRT GPU sobre el mismo modelo. Reportar latencia P50 y P99.

Homework verificable

Servir un modelo de Fashion-MNIST con TF Serving y comparar con ONNX:

1. model.export('servable/1/').
2. Docker run TF Serving.
3. Convert TF → ONNX.
4. Inference con ambos sobre los mismos 100 inputs; verificar outputs idénticos (±1e-5).
5. Comparar latencia.

Criterio de aceptación: predicciones de TF Serving y ONNX Runtime coinciden; latencia reportada para ambos.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
TF Serving no encuentra modelo	Estructura de carpetas mal. Fix: model_nam

Outputs cambian entre TF y ONNX	Conversión incompleta (capas custom). Fix:
TensorRT engine no portable entre GPUs	Es específico al hardware. Fix: rebuild en
vLLM no carga modelo cuantizado	Algunas quantizations no soportadas. Fix:
Latencia P99 alta	Sin warm-up. Fix: warmup_count=10 en TF Se

Preguntas frecuentes

¿REST o gRPC?

gRPC para producción serio (más rápido). REST para debugging y compatibilidad universal.

¿ONNX es siempre más rápido que TF Serving?

No siempre. ONNX brilla en CPU + casos específicos. TF Serving está más optimizado para batching en GPU.

¿TensorRT necesario?

Solo si latencia es crítica. Para 99 % de los casos, ONNX Runtime + GPU es suficiente.

¿Cómo manejo versioning de modelos?

TF Serving por default sirve el "latest" entre las carpetas numeradas. Para A/B: serve varias versiones, configurar weights en model_config_list.

¿LLM en TF Serving o vLLM?

vLLM siempre para LLMs. TF Serving es ineficiente para autoregresivo.

Referencias

- Géron, cap. 19 — Deploying a Model to a Service.
- TF Serving docs.
- ONNX docs, ONNX Runtime.
- Triton Inference Server.
- vLLM docs, TGI docs.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 167 — Clase 167 — ONNX y ONNX Runtime: portabilidad e inference optimizada

> *Parte: 2 — Deep Learning · Fuente: ONNX docs + ONNX Runtime team blogs. Duración estimada: 80 min.*

Objetivo

Dominar ONNX (formato intermedio) y ONNX Runtime (runtime cross-platform) — la solución portable para inference: entrenás en TF/PyTorch/JAX, exportás a ONNX, corrés en cualquier hardware (CPU, GPU NVIDIA, GPU AMD, mobile, browser, edge). Conocer TensorRT (NVIDIA-specific, mayor performance).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Exportar modelos: torch.onnx.export, tf2onnx.convert.
- Cargar e inferir con onnxruntime.InferenceSession.
- Elegir execution provider: CPU, CUDA, TensorRT, OpenVINO, DirectML, CoreML.
- Optimizar modelos: graph optimization, quantization int8/fp16.
- Reconocer cuándo ONNX vs TF Serving vs vLLM (LLMs).

Temas

- ONNX como protocolo (protobuf): operator set + tensor types.
- Conversión: cada framework tiene su exporter.
- Verificación: outputs deben coincidir framework ↔ ONNX (±1e-5).
- Optimization: graph simplification, layer fusion.
- Quantization: dynamic, static (con calibration), QAT.
- Execution providers: priority order.

Definiciones y características

- ONNX: formato; archivo .onnx.
- Opset version: versión del set de ops. Higher = newer ops (ej. opset 17 incluye attention).
- InferenceSession: objeto runtime que carga modelo y ejecuta.
- Execution Provider (EP): backend de hardware. ORT elige el primero disponible.
- onnxruntime-gpu: pip package específico para CUDA.
- onnxruntime-directml: Windows GPU (AMD/Intel via DirectX).

Dataset / recursos

- Modelo de clases anteriores (Fashion-MNIST o ResNet preentrenado).
- Librerías: onnx, onnxruntime o onnxruntime-gpu, tf2onnx (TF), torch.onnx built-in.

Ejercicios

1. PyTorch → ONNX: torch.onnx.export(model, dummy_input, 'model.onnx', opset_version=17, input_names=['input'], output_names=['output']).
2. TF → ONNX: python -m tf2onnx.convert --saved-model dir/ --output model.onnx --opset 17.
3. Inference: sess = ort.InferenceSession('model.onnx', providers=['CUDAExecutionProvider', 'CPUExecutionProvider']). Verificar outputs.
4. Graph optimization: sess_options.graph_optimization_level = ort.GraphOptimizationLevel.ORT_ENABLE_ALL.
5. Quantization: onnxruntime.quantization.quantize_dynamic('model.onnx', 'model_q.onnx', weight_type=QuantType.QUInt8).

Homework verificable

Exportar y deployar un modelo CNN ya entrenado:

1. PyTorch ResNet50 preentrenado → ONNX.
2. Quantization dynamic.
3. Comparar latencia + accuracy: PyTorch CUDA vs ORT CUDA vs ORT CPU vs ORT CPU quantized.
4. Verificar correctness (output diff < 1e-4 vs PyTorch).

Criterio de aceptación: ORT CUDA ≥ PyTorch CUDA en velocidad; quantization reduce tamaño 4× con < 1 pp accuracy loss.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
opset version X not supported	EP version vieja. Fix: actualizar onnxrunt
Output diff vs framework grande	Conversión incompleta (custom ops). Fix: v
CUDA provider no disponible	Falta onnxruntime-gpu install. Fix: pip in
Dynamic axes mal	Mal export. Fix: dynamic_axes={'input': {0
Quantization rompe accuracy	Modelo sensible. Fix: usar static quantiza

Preguntas frecuentes

ONNX vs TensorRT?

ONNX Runtime + TensorRT EP combina ambos: ORT como interfaz, TensorRT como backend para GPUs NVIDIA. Best of both.

ONNX en mobile?

ONNX Runtime Mobile: optimizado para Android/iOS. Soporta CoreML, NNAPI delegates.

ONNX en browser?

ONNX.js — corre en WebGL/WebGPU/WASM.

LLMs con ONNX?

Posible (especialmente con onnxruntime-genai). vLLM/TGI siguen siendo mejores para LLMs grandes.

Triton vs ORT?

Triton (NVIDIA) puede servir modelos ONNX como uno de sus backends. Triton para infra multi-modelo; ORT solo para inference.

Referencias

- ONNX.
- ONNX Runtime.
- ONNX Runtime Mobile.
- ONNX Runtime GenAI.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 168 — Clase 168 — Despliegue en Vertex AI

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 19 § Deploying a Model to Vertex AI + docs Vertex AI. Duración estimada: 60 min.*

Objetivo

Desplegar un modelo a Vertex AI (GCP) — el servicio managed de Google para servir modelos sin maintener

infrastructure. Conocer alternativas: AWS SageMaker, Azure ML, Modal, Replicate, HuggingFace Inference Endpoints.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Subir un modelo a Vertex AI Model Registry.
- Crear un Endpoint y deployar el modelo.
- Hacer requests a un endpoint Vertex AI.
- Comparar managed (Vertex/SageMaker) vs self-hosted (TF Serving + GKE/EKS).
- Conocer alternativas modernas (Modal, Replicate) para deploy serverless.

Temas

- Vertex AI Model Registry.
- Endpoint creation + traffic split (A/B testing).
- gcloud CLI: gcloud ai models upload, gcloud ai endpoints deploy-model.
- Pricing: pay per CPU/GPU hour + requests.
- Alternativas: SageMaker, Azure ML, Modal, Replicate.

Definiciones y características

- Model Registry: catálogo de modelos versionados.
- Endpoint: URL HTTP para inference.
- Traffic split: routing parcial entre versiones (A/B).
- Auto-scaling: replicas automáticas según carga.
- Prebuilt containers: TF/PyTorch/sklearn containers oficiales.

Dataset / recursos

- Modelo de clases previas exportado.
- GCP account con billing habilitado (free tier limitado).
- Librerías: google-cloud-aiplatform.

Ejercicios

1. Setup: gcloud init, gcloud auth application-default login. Crear bucket GCS.
2. Upload model: gcloud ai models upload --display-name=fashion --container-image-uri=....
3. Deploy endpoint: con n1-standard-4. Min/max replicas 1-3.
4. Predict request: from google.cloud import aiplatform; ep = aiplatform.Endpoint(...); ep.predict(...).
5. A/B traffic: deploy v2 con 20 % traffic, observar logs.

Homework verificable

Deploy del modelo Fashion-MNIST a Vertex AI:

1. Subir a GCS.
2. Upload model en Vertex Model Registry.
3. Crear endpoint y deploy.
4. Hacer 10 predicciones desde notebook local.
5. (Opcional) cleanup para no gastar.

Criterio de aceptación: predicciones llegan correctamente; cost report < $1.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
PERMISSION_DENIED al deploy	IAM mal configurado. Fix: rol aiplatform.a
Endpoint queda corriendo y factura $$$	Olvidaste undeploy. Fix: gcloud ai endpoin
Modelo no carga	Formato no soportado. Fix: SavedModel form
Predicciones lentas (cold start)	Auto-scale a 0 y arranque toma 30-60s. Fix
Output incomprensible	Signature mal definida. Fix: documentar in

Preguntas frecuentes

¿Vertex AI o SageMaker?

Depende del ecosistema. Si ya estás en AWS → SageMaker. En GCP → Vertex. Funcionalmente similares.

¿Modal / Replicate cuándo?

Para deployment rápido (serverless, pay-per-request) y para LLMs/diffusion específicamente. Modal es excelente DX, Replicate tiene modelos pre-built.

¿Self-host con K8s en lugar de managed?

Si tenés equipo de ops y volumen alto, sale más barato. Para empezar o equipos chicos, managed gana.

¿Costos típicos?

Endpoint con n1-standard-4 24/7 ≈ $100/mes. Con GPU T4 ≈ $250/mes. Si auto-scale a 0 entre requests, mucho menos.

¿Inferencia batch para datasets grandes?

gcloud ai batch-predict-jobs create — más barato que endpoint para volúmenes grandes sin necesidad real-time.

Referencias

- Géron, cap. 19 — Deploying a Model to Vertex AI.
- Vertex AI docs.
- SageMaker docs.
- Modal, Replicate.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 169 — Clase 169 — TF Lite (mobile/embedded)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 19 § Deploying a Model to a Mobile or Embedded Device. Duración estimada: 55 min.*

Objetivo

Convertir y desplegar un modelo a TensorFlow Lite (LiteRT) — runtime optimizado para móviles (Android/iOS), IoT y embedded (Raspberry Pi, microcontroladores). Aplicar quantization (int8) para reducir

tamaño 4× y acelerar 2-4× en CPU móvil.

> *Nota: TF Lite fue renombrado a LiteRT en 2024 (mismo proyecto, nuevo branding bajo el paraguas de "AI Edge").*

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Convertir SavedModel a .tflite: converter = tf.lite.TFLiteConverter.from_saved_model('servable/1/'); tflite_model = converter.convert().
- Aplicar quantization post-training (dynamic range, int8 full).
- Cargar y ejecutar inference con tf.lite.Interpreter.
- Conocer alternativas modernas: ONNX Runtime Mobile, CoreML (iOS), NNAPI (Android).
- Reconocer trade-off accuracy vs tamaño/velocidad.

Temas

- TF Lite vs TF: subset de ops, runtime minimal.
- Conversión SavedModel → tflite.
- Quantization types: dynamic range (weights int8), int8 full (weights + activations), float16.
- tf.lite.Interpreter API.
- Mobile delegates: NNAPI, GPU, CoreML.

Definiciones y características

- TF Lite: runtime para edge devices.
- Quantization: convertir float32 → int8/float16 → modelo más chico y rápido.
- Calibration dataset: ~100 ejemplos para calibrar quantization de activaciones.
- Delegate: hardware-specific accelerator (NNAPI en Android, CoreML en iOS, Edge TPU).
- .tflite: archivo binario compacto.

Dataset / recursos

- Modelo Fashion-MNIST.
- Librerías: tensorflow.

Ejercicios

1. Convert: TFLiteConverter.from_saved_model(...).convert() → guardar .tflite. Comparar tamaño vs SavedModel.
2. Inference: cargar .tflite y hacer predict con Interpreter. Verificar misma predicción que el modelo original.
3. Dynamic range quant: converter.optimizations = [tf.lite.Optimize.DEFAULT]. Tamaño 4× menor.
4. Full int8: definir representative_dataset y converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]. Comparar tamaño y accuracy.
5. Latencia: medir tiempo de inference en CPU laptop. Comparar versión float32 vs int8.

Homework verificable

Pipeline de Fashion-MNIST a TF Lite:

1. Convertir + 3 quantizations (none, dynamic, int8 full).
2. Para cada uno: tamaño + accuracy en test + latencia.
3. Reportar tabla comparativa.

Criterio de aceptación: int8 reduce tamaño ~4× con < 1 pp accuracy loss; latencia 2-3× más rápida.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
op not supported al convertir	Algunas ops Keras no son TFLite-compatible
int8 full quant con representative dataset	Mala calibración. Fix: ≥ 100 ejemplos dive
Tamaño del .tflite no baja con quant	El modelo era pequeño. Quantization brilla
Inference más lenta de lo esperado	Sin delegate. Fix: NNAPI (Android), CoreML
Predicciones distintas float vs int8	Esperable; verificar accuracy en test set.

Preguntas frecuentes

¿TFLite o ONNX Runtime Mobile?

TFLite si entrenas en TF; ONNX para portabilidad multi-framework. Funcionalmente comparable.

¿iOS — TFLite o CoreML?

CoreML mejor integración con iOS (Neural Engine de Apple). Convertir TFLite → CoreML con coremltools.

¿Microcontroladores ARM Cortex-M?

TF Lite Micro (TFLM) o MicroPython TF — para inference en MCU con KB de RAM. Modelos diminutos (< 100 KB).

¿Quantization durante training?

QAT (Quantization-Aware Training) — entrenar simulando los efectos de int8. Mejor accuracy que post-training quant.

¿Cuándo NO desplegar a mobile?

Modelos grandes (>100 MB) o que requieren GPU server (LLMs grandes). Para LLMs en mobile, ver MLC LLM o llama.cpp con quantization GGUF.

Referencias

- Géron, cap. 19 — Deploying a Model to a Mobile or Embedded Device.
- TensorFlow Lite (LiteRT) docs.
- ONNX Runtime Mobile.
- CoreML Tools.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 170 — Clase 170 — TensorFlow.js (navegador)

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 19 § Running a Model in a Web Page + TF.js docs. Duración estimada: 55 min.*

Objetivo

Servir modelos directamente en el navegador con TensorFlow.js — corre client-side (privacidad, sin server cost, sin latency network). Alternativas modernas: ONNX Runtime Web, WebGPU-accelerated inference, transformers.js (Hugging Face) para LLMs en el browser.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Convertir un Keras model a TF.js format con tensorflowjs_converter.
- Cargar y hacer inference desde JavaScript en el browser.
- Conocer WebGL backend (default TF.js) y WebGPU (nuevo, más rápido).
- Usar transformers.js para correr modelos NLP/visión en browser.
- Reconocer cuándo conviene client-side vs server-side.

Temas

- Conversion: tensorflowjs_converter --input_format=keras model.keras tfjs_model/.
- JS API: const model = await tf.loadLayersModel('tfjs_model/model.json').
- Backends: WebGL (default), WASM, WebGPU.
- transformers.js: ONNX en browser, soporta BERT, GPT-2, Whisper.
- Edge cases: tamaño del modelo (~5-50 MB ideal), latencia primer load.

Definiciones y características

- TF.js: implementación de TF en JavaScript.
- WebGL backend: usa GPU del browser, default.
- WebGPU: API moderna, 2-5× más rápida que WebGL.
- WASM backend: CPU-only pero portable.
- transformers.js: HuggingFace en browser, ONNX-based.

Dataset / recursos

- Modelo Fashion-MNIST.
- Librerías: tensorflowjs, tensorflowjs-converter.

Ejercicios

1. Convert: tensorflowjs_converter --input_format=keras_saved_model servable/ tfjs_model/. Inspeccionar archivos.
2. HTML page: pagina simple que carga model.json, dibuja una imagen 28×28 en canvas, predice.
3. WebGPU: await tf.setBackend('webgpu'). Comparar velocidad vs WebGL.
4. transformers.js: import { pipeline } from '@xenova/transformers'. Correr sentiment-analysis en browser. 1 línea.
5. PWA: empaquetar como Progressive Web App con service worker para offline inference.

Homework verificable

Demo de Fashion-MNIST en browser:

1. Convertir modelo a TF.js.
2. Página HTML con canvas donde el usuario dibuja.
3. Botón "Predict" → muestra clase + probabilidades.

Criterio de aceptación: la demo funciona en Chrome/Firefox; predicciones son razonables para dibujos sencillos.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
model.json not found	Path mal o servidor no sirve archivos está
OOM en browser con modelo grande	TF.js no es para GB-scale models. Fix: mod
WebGL fail en algunos browsers	Falta WebGL2. Fix: fallback a WASM.
Predicciones diferentes vs server	Diferencia en preprocesamiento. Fix: hacer
Modelo carga lento	Sin caching. Fix: Service Worker + cache.

Preguntas frecuentes

¿TF.js o ONNX Runtime Web?

TF.js para modelos TF; ORT Web para portabilidad. ORT Web también soporta WebGPU.

¿LLMs en browser?

Sí — modelos chicos (DistilBERT, TinyLlama, Whisper tiny) con quantization. WebLLM proyecto usa MLC y WebGPU para Llama 2 7B en browser.

¿Privacidad?

Client-side = data nunca sale del navegador. Ideal para healthcare, finanzas.

¿WebAssembly (WASM) vs WebGL?

WASM es CPU portable, lento. WebGL usa GPU. WebGPU es el futuro (5× WebGL).

¿Cuándo NO usar client-side?

Modelos grandes (>100 MB), datasets propietarios (no quieres exponerlos), latencia inaceptable en primer load.

Referencias

- Géron, cap. 19 — Running a Model in a Web Page.
- TensorFlow.js docs.
- transformers.js.
- ONNX Runtime Web.
- MLC WebLLM.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 171 — Clase 171 — Aceleración con GPU

> *Parte: 2 — Deep Learning · Fuente: Géron, cap. 19 § Using GPUs to Speed Up Computations. Duración estimada: 60 min.*

Objetivo

Configurar GPU para DL: drivers, CUDA, cuDNN, verificación. Conocer mixed precision (bfloat16/float16) que duplica throughput en GPUs modernas (Ampere/Hopper). Profilear con TensorBoard Profiler para identificar

bottlenecks (data loading vs compute vs sync).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Verificar GPU: tf.config.list_physical_devices('GPU'), torch.cuda.is_available().
- Activar mixed precision: keras.mixed_precision.set_global_policy('mixed_float16') (Volta+) o 'mixed_bfloat16' (Ampere+).
- Limitar memoria GPU: set_memory_growth(True) para no consumir toda al inicio.
- Multi-GPU básico con tf.distribute.MirroredStrategy (clase 144 profundiza).
- Profilear con TensorBoard Profiler.

Temas

- CUDA toolkit + cuDNN versions matching.
- nvidia-smi para monitor.
- Mixed precision: float16 (Volta+, 16 GB max) vs bfloat16 (Ampere+, mismo exponente que float32, más estable).
- Memory growth vs allocated all.
- Profiling con TensorBoard.
- GPUs típicos en 2026: H100, H200 (server); RTX 5090 (consumer).

Definiciones y características

- CUDA: API de NVIDIA para GPU compute.
- cuDNN: librería de primitives DL (Conv, RNN, MatMul).
- Mixed precision: usar float16/bfloat16 para forward+backward, float32 para weights master copy.
- mixed_float16: requiere loss scaling para evitar underflow.
- mixed_bfloat16: mismo rango que float32, no necesita scaling.
- TPU: ASIC de Google. Diferente API (XLA).

Dataset / recursos

- Modelo + dataset cualquier de las clases previas.
- Librerías: tensorflow, opcional nvidia-smi.

Ejercicios

1. GPU check: imprimir tf.config.list_physical_devices('GPU'), tf.config.list_logical_devices('GPU'), nvidia-smi.
2. Memory growth: tf.config.experimental.set_memory_growth(gpu, True) para evitar reservar 100 % al inicio.
3. Mixed precision: keras.mixed_precision.set_global_policy('mixed_float16'). Re-entrenar modelo. Verificar speedup (1.5-2× en V100; 2-3× en A100/H100).
4. Profile: keras.callbacks.TensorBoard(log_dir=..., profile_batch=(5, 10)). Abrir Profiler tab.
5. Bottleneck: si la GPU está al 30 %, el bottleneck es data loading. Optimizar pipeline (clase 109).

Homework verificable

Optimizar el training de un modelo:

1. Baseline con default config.
2. Activar mixed precision + memory_growth.
3. Profile y identificar bottleneck.
4. Aplicar fix (más prefetch, batch más grande, etc.).

5. Reportar speedup wall-clock.

Criterio de aceptación: speedup ≥ 1.5× con mixed precision; identificado el bottleneck correctamente.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Could not find cuda drivers	Drivers no instalados o versión incompatib
OOM al usar mixed precision	A veces el ahorro de memoria no es total.
mixed_float16 loss = NaN	Loss scaling automático debe estar activo.
GPU al 100 % CPU al 100 % también	Data loading no es el bottleneck → OK. Si
Múltiples GPU pero usa 1	No configuraste strategy. Fix: MirroredStr

Preguntas frecuentes

¿float16 o bfloat16?

bfloat16 si GPU lo soporta (Ampere+, A100/H100, TPU). Más estable, sin loss scaling. float16 en GPUs viejas (V100, T4).

¿GPU consumer (RTX) para DL profesional?

Sí — RTX 4090 / 5090 tienen 24 GB VRAM y excelente performance. Para LLMs grandes (>30B), considerar A100/H100.

¿Cuántas GPU necesito?

1 GPU para experimentos. Multi-GPU para datasets grandes (ImageNet) o modelos grandes (LLMs).

¿Apple Silicon (M-chips)?

TF tiene soporte vía tensorflow-metal. PyTorch via MPS backend. Para experimentos chicos OK; producción serio sigue siendo NVIDIA o TPU.

¿GPU en cloud?

AWS p4/p5, GCP A2/A3, Lambda Labs, Vast.ai. Lambda/Vast son más baratos para experimentos.

Referencias

- Géron, cap. 19 — Using GPUs to Speed Up Computations.
- TF GPU guide.
- Mixed precision guide.
- NVIDIA H100 / Hopper architecture whitepaper.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 172 — Clase 172 — Entrenamiento multi-dispositivo, tf.distribute

Parte: 2 — Deep Learning · Fuente: Géron, cap. 19 § Training Models Across Multiple Devices.

Duración estimada: 75 min.

Objetivo

Escalar el training a múltiples GPUs (y multi-nodos). Conocer las 3 estrategias TF: MirroredStrategy (1 nodo, varias GPUs), MultiWorkerMirroredStrategy (varios nodos), TPUStrategy. Conocer equivalentes PyTorch (DDP, FSDP) que son estándar para LLMs grandes.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar MirroredStrategy en un único nodo con N GPUs.
- Entender data parallelism: cada GPU procesa su mini-batch, gradients se promedian (all-reduce).
- Diferenciar de model parallelism (modelo dividido entre GPUs) y pipeline parallelism.
- Conocer FSDP (Fully Sharded Data Parallel) para modelos demasiado grandes para 1 GPU (LLMs).
- Saber que PyTorch Lightning abstrae todo esto cambiando un kwarg.

Temas

- Data parallelism: misma red replicada, distinto batch por GPU.
- Model parallelism: red dividida (tensor parallel, pipeline parallel).
- FSDP / DeepSpeed ZeRO: shard de parámetros, gradientes, optimizer states.
- TF: MirroredStrategy, MultiWorkerMirroredStrategy, TPUStrategy.
- PyTorch: DistributedDataParallel, FullyShardedDataParallel, DeepSpeed.
- Lightning / Accelerate: abstracciones de alto nivel.

Definiciones y características

- Data Parallelism: cada device procesa un slice del batch.
- All-reduce: operación colectiva que promedia gradients entre devices.
- MirroredStrategy: data parallelism single-node.
- MultiWorkerMirroredStrategy: data parallelism multi-node.
- FSDP: shards de parámetros entre devices — para modelos demasiado grandes.
- Gradient accumulation: simular batch grande acumulando gradients en GPUs chicas.

Dataset / recursos

- Acceso a ≥ 2 GPUs (Colab Pro+, cloud).
- Librerías: tensorflow, opcional torch.distributed.

Ejercicios

1. MirroredStrategy: strategy = tf.distribute.MirroredStrategy(). with strategy.scope(): model = build_model(); model.compile(...). Entrenar.
2. Batch effective: si tenés 4 GPUs y batch_size=32, el batch global es 128. Adjust LR (LR scales linearly with batch).
3. Gradient accumulation: simular batch=512 acumulando 4 mini-batches de 128. Manual con custom training loop.
4. PyTorch DDP: comando torchrun --nproc_per_node=4 train.py con DistributedDataParallel(model).
5. PyTorch Lightning: trainer = L.Trainer(strategy='ddp', devices=4). Listo.

Homework verificable

Si tenés ≥ 2 GPUs:

1. Mismo modelo en single-GPU vs MirroredStrategy(2 GPUs).
2. Reportar wall-time por epoch.
3. Verificar que la accuracy final es similar.

Criterio de aceptación: speedup wall-clock ≈ 1.5-1.9× con 2 GPUs (no es 2× porque hay overhead de comm); accuracy similar.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
MirroredStrategy no acelera	Bottleneck en data loading. Fix: optimizar
LR no ajustado con batch global mayor	Convergencia subóptima. Fix: lr_new = lr_o
OOM en multi-GPU con LLM grande	Data parallelism replica el modelo. Si el
All-reduce muy lento	Network bandwidth limitado en multi-nodo.
Mismatch en BatchNorm stats entre replicas	Sync BN: tf.keras.layers.experimental.Sync

Preguntas frecuentes

¿Multi-GPU cuándo necesario?

Dataset masivo (días en 1 GPU) o modelo grande (no entra en 1 GPU). Para experimentos chicos, 1 GPU.

¿FSDP o DeepSpeed?

Ambas implementan shard de pesos. FSDP es PyTorch nativo. DeepSpeed (Microsoft) tiene más features (ZeRO-Offload, CPU offloading).

¿Train LLM 70B en 1 nodo?

8× H100 (640 GB VRAM total) con FSDP es factible para Llama 70B. Cluster necesario para más grande.

¿Cloud para multi-GPU?

AWS p4d (8× A100), GCP A3 (8× H100). Caro ($30-60/hora).

¿MultiWorkerMirroredStrategy real-world?

Sí, con TF_CONFIG env var y orquestación (K8s, Slurm, Vertex AI). Vertex AI hace esto transparente.

Referencias

- Géron, cap. 19 — Training Models Across Multiple Devices.
- tf.distribute guide.
- PyTorch DDP, FSDP.
- DeepSpeed.
- Rajbhandari et al. (2020), ZeRO: Memory Optimizations Toward Training Trillion Parameter Models.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 173 — Clase 173 — JAX y Flax: el stack moderno de Google para DL

Parte: 2 — Deep Learning · Fuente: JAX docs + Flax NNX docs. Duración estimada: 85 min.

Objetivo

Aprender JAX (Google 2018) y Flax (NN library on top of JAX) — el stack que sostiene AlphaFold, Gemini, MaxText, AlphaCode y muchos modelos modernos. Cubrir jit, vmap, pmap, grad, transformaciones funcionales, y Flax NNX (la nueva API 2024, similar a PyTorch).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diferenciar JAX (NumPy + autodiff + XLA) de NumPy plano.
- Usar jax.jit para compilación XLA (2-10× speedup automático).
- Aplicar jax.vmap (vectorización automática) y jax.grad (autodiff funcional).
- Construir modelos con Flax NNX (API moderna similar a PyTorch).
- Reconocer cuándo elegir JAX sobre PyTorch (TPU, escala extrema, research).

Temas

- Functional programming: funciones puras, no mutación.
- jit, grad, vmap, pmap como transformaciones.
- XLA: compilación a hardware específico (CPU, GPU, TPU).
- PRNG explícito (jax.random.PRNGKey).
- Flax: NN library. Antes Linen (functional), ahora NNX (stateful, más parecido a PyTorch).
- Optax: optimizadores.

Definiciones y características

- jax.numpy: drop-in NumPy con autodiff + XLA.
- jit: compila la función a XLA. Primera vez lento, después rápido.
- grad: devuelve función que computa gradiente.
- vmap: vectoriza sobre un eje. Como agregar batch dim.
- pmap: paraleliza sobre dispositivos (multi-GPU/TPU).
- PRNGKey: random determinista. jax.random.split(key) para usar varias veces.
- NNX: API stateful de Flax 2024, reemplaza Linen.

Dataset / recursos

- Fashion-MNIST.
- Librerías: jax, jaxlib, flax, optax.

Ejercicios

1. JAX basic: import jax.numpy as jnp; x = jnp.array([1.,2.,3.]); jnp.sum(x**2). Comparar contra NumPy.
2. grad: grad_f = jax.grad(lambda x: x**3); grad_f(2.) → 12.
3. jit speedup: definir función numérica, medir tiempo con y sin @jax.jit. ≥ 5× speedup.
4. vmap: función para una muestra → vmap para procesar batch.
5. Flax NNX MLP: definir modelo, training step, entrenar Fashion-MNIST.

Homework verificable

Re-entrenar Fashion-MNIST en JAX/Flax:

1. Modelo Flax NNX con 2 capas Dense.
2. Optax adam(1e-3).

3. Training loop con jit.
4. Reportar accuracy + tiempo vs equivalente PyTorch.

Criterio de aceptación: accuracy ≥ 0.87; JIT activo (segunda llamada mucho más rápida que primera).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Primera ejecución muy lenta	Compilación XLA. Fix: paciencia, después e
TracerError	Mutación dentro de jit. Fix: pure function
OOM en TPU	Modelo grande sin sharding. Fix: pmap o pj
Random no determinista	Olvidaste split del PRNGKey. Fix: key, sub
Comparar PyTorch vs JAX wall-clock sin JIT	Sin JIT, JAX es lento. Fix: siempre con @j

Preguntas frecuentes

JAX o PyTorch?

PyTorch ecosystem es más grande. JAX brilla en research / TPU / extrema escala. Para 99 % de DL aplicado, PyTorch.

Linen o NNX?

NNX (2024) — más fácil, stateful, parecido a PyTorch. Linen es legacy.

TPU en cloud?

Google Cloud TPU v4/v5e/v5p. Vertex AI lo wrappea. Caro pero performance excelente para batch grande.

Hugging Face soporta JAX?

Sí, muchos modelos tienen versión JAX. Pero la mayoría de la actividad es PyTorch.

¿AlphaFold en JAX?

Sí. JAX brilla en numerical computing (física, chemistry, biology).

Referencias

- JAX docs.
- Flax docs.
- Optax docs.
- Bradbury et al. (2018), JAX: composable transformations of Python+NumPy programs.
- MaxText — LLM training en JAX.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 174 — Clase 174 — Entrenamiento a escala con Vertex AI

Parte: 2 — Deep Learning · Fuente: Géron, cap. 19 § Running Large Training Jobs on Vertex AI + docs

Vertex AI Training. Duración estimada: 65 min.

Objetivo

Cierre del bloque de despliegue: lanzar training jobs a escala en Vertex AI (GCP managed) — cluster automático, GPUs/TPUs on-demand, hyperparameter tuning distribuido. Conocer alternativas: AWS SageMaker Training, Azure ML Jobs, y plataformas dedicadas a LLMs (Modal, Together AI).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Empaquetar un training script como Docker container.
- Lanzar un Custom Training Job en Vertex AI con gcloud ai custom-jobs create.
- Configurar HP tuning con Vertex AI Vizier.
- Usar TPU Pods para training distribuido extremo.
- Conocer alternativas cloud-agnostic (Modal, Together, RunPod).

Temas

- Custom container vs prebuilt container en Vertex.
- WorkerPoolSpec: master + workers + parameter servers.
- TPU pods para training a escala.
- Hyperparameter tuning con Vizier (Bayesian search).
- Costos: GPU/TPU hours.
- Alternativas: SageMaker, Modal, Together, RunPod, Lambda Labs.

Definiciones y características

- Custom Training Job: corre tu container en Vertex.
- WorkerPool: cluster spec (machine_type, count, accelerator).
- Vizier: black-box optimizer integrado para HP tuning.
- TPU: tensor processing unit, ASIC de Google.
- Spot instances / preemptible: ~70 % más barato pero pueden ser killed.

Dataset / recursos

- GCP account.
- Librerías: google-cloud-aiplatform.

Ejercicios

1. Dockerize: escribir Dockerfile con TF + tu training script.
2. Lanzar job: aiplatform.CustomJob(display_name='exp1', worker_pool_specs=[...]).run().
3. HP tuning: HyperparameterTuningJob con Vizier; 20 trials.
4. Multi-GPU spec: machine_type='a2-highgpu-4g' (4× A100).
5. TensorBoard integration: Vertex TB para monitor.

Homework verificable

Lanzar un training job de Fashion-MNIST en Vertex:

1. Containerizar.
2. Job de 1 GPU (n1-standard-4 + T4).
3. Logs y output a GCS.
4. Verificar accuracy similar a entrenamiento local.

Criterio de aceptación: el job termina con accuracy ≥ 0.87 y el modelo queda guardado en GCS.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Job falla al iniciar	Image no accesible. Fix: push a Artifact R
Olvidé apagar y factura $1000	Vertex jobs son time-limited pero HP tunin
Logs no aparecen	Tarda 1-2 min en stream. Fix: paciencia o
Multi-GPU pero job usa 1	TF no detecta GPUs en el container. Fix: i
Spot interrumpe a mitad	Aceptable para experimentos. Fix: checkpoi

Preguntas frecuentes

¿Cuándo Vertex vs local?

Local para iteración. Cloud cuando: dataset no entra en RAM local, training > 1 día, HP tuning con muchos trials, multi-GPU/TPU.

¿TPU vs GPU?

TPU brilla en modelos TF/JAX grandes (LLMs, transformers de visión). GPU es más universal. PyTorch en TPU funciona (vía XLA) pero menos óptimo.

¿Spot/preemptible?

Para training con checkpointing, ahorra 70 %. Para inference o jobs cortos, evitar.

¿Modal vs Vertex?

Modal: DX excelente, billing por segundo, ideal para LLMs e inference serverless. Vertex: más enterprise features (compliance, IAM, audit).

¿Together AI / Replicate cuándo?

Cuando solo necesitás API a modelos open-source preentrenados (LLMs, difusión). No para training desde cero.

Referencias

- Géron, cap. 19 — Running Large Training Jobs on Vertex AI.
- Vertex AI Training docs.
- SageMaker Training.
- Modal, Together AI, RunPod.

Siguiente parte

Clase 175 — Distribuciones: normal, binomial, Poisson, exponencial

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Cierre de la parte

Fin del bundle consolidado de Parte 2 — Deep Learning — Keras, TensorFlow, Transformers, RL y Despliegue · 75 clases.