DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Machine Learning Operations (MLOps) Clase 4

Leticia Rodríguez

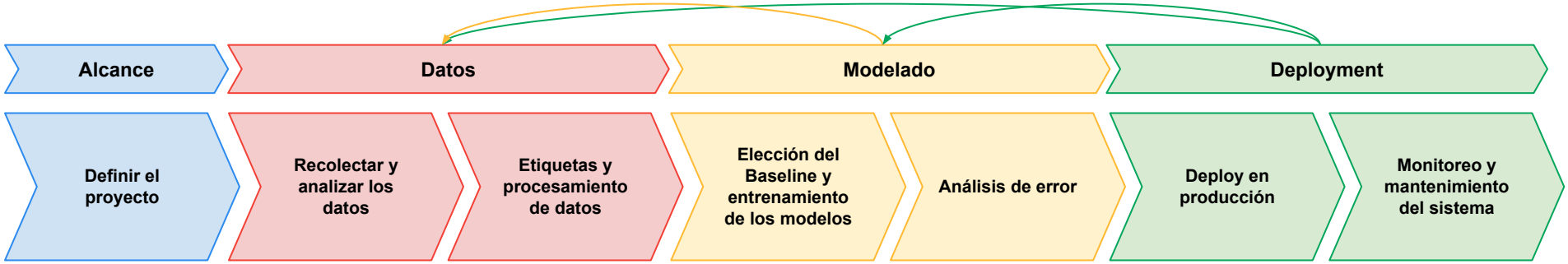
Septiembre 2024 - 2do Cuatrimestre - 4to. Bimestre

Universidad de Buenos Aires - FCEyN - Departamento de Computación

Formulario y Asistencia

Kahoot de respaso

El flujo de creación de un modelo de ML

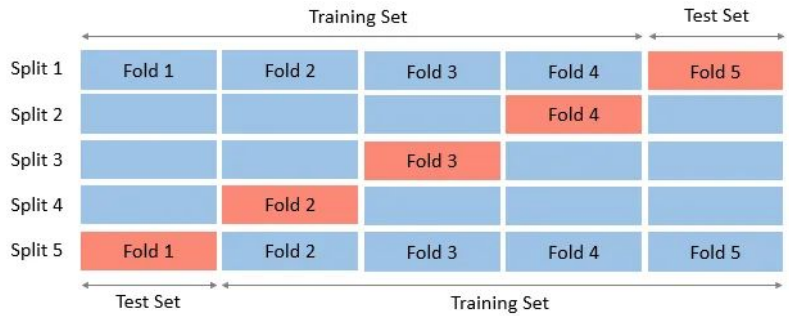


Importante: las diferentes etapas tienen flechas que hace que se itere entre las etapas anteriores

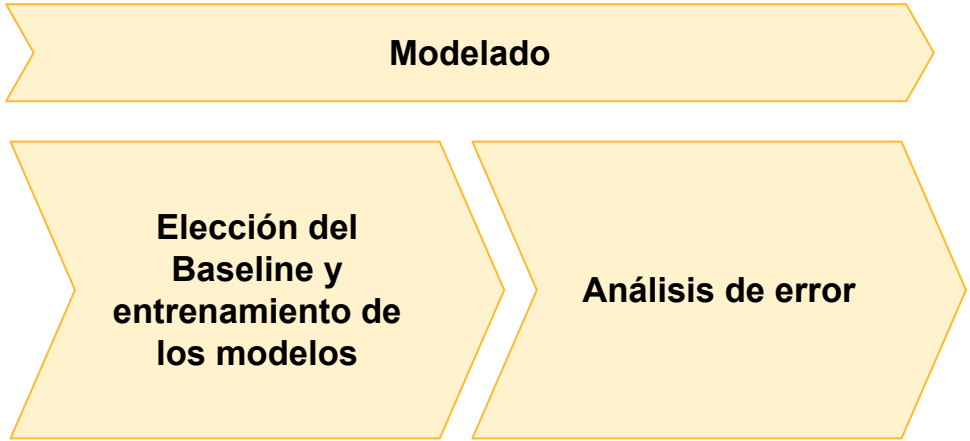
Datos de Entrenamiento, Validación, y Testeo

- Para la preparación del modelo se suelen precisar 3 datasets: entrenamiento, validación y testeo.
- Elegidos de forma que reflejan la variedad de datos que va a recibir el modelo en predicción
- El tamaño debe ser representativo de la performance del modelo.
- Ideas iniciales:
  - Muchos Datos:
    - 120.000 imágenes - 80% Entrenamiento 20% Testeo
    - Elegir un subset de datos y ahi partir en 3 sets: 70% train, 15% val, 15% test
  - Pocos Datos:
    - Técnicas para generar datos o buscar más datos
    - k-fold cross validation

1. Entrenar el modelo usando k-1 folds
2. Testear el modelo en los todos los folds excepto el usado para validación durante entrenamiento
3. La performance del modelo es el promedio de los resultados

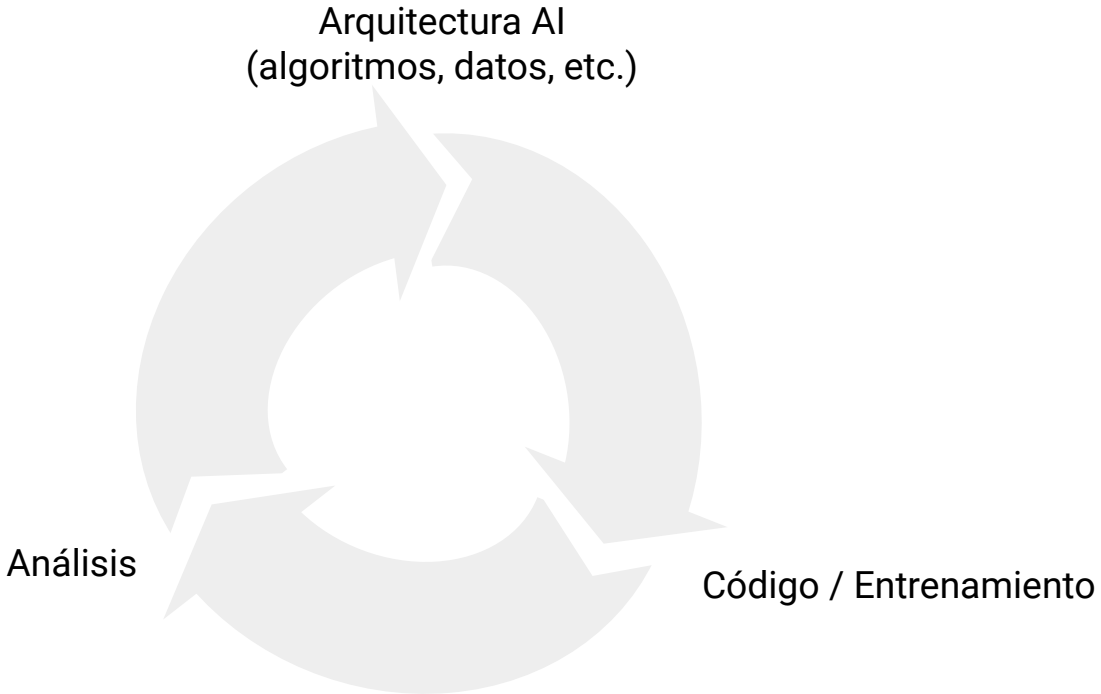


Modelado

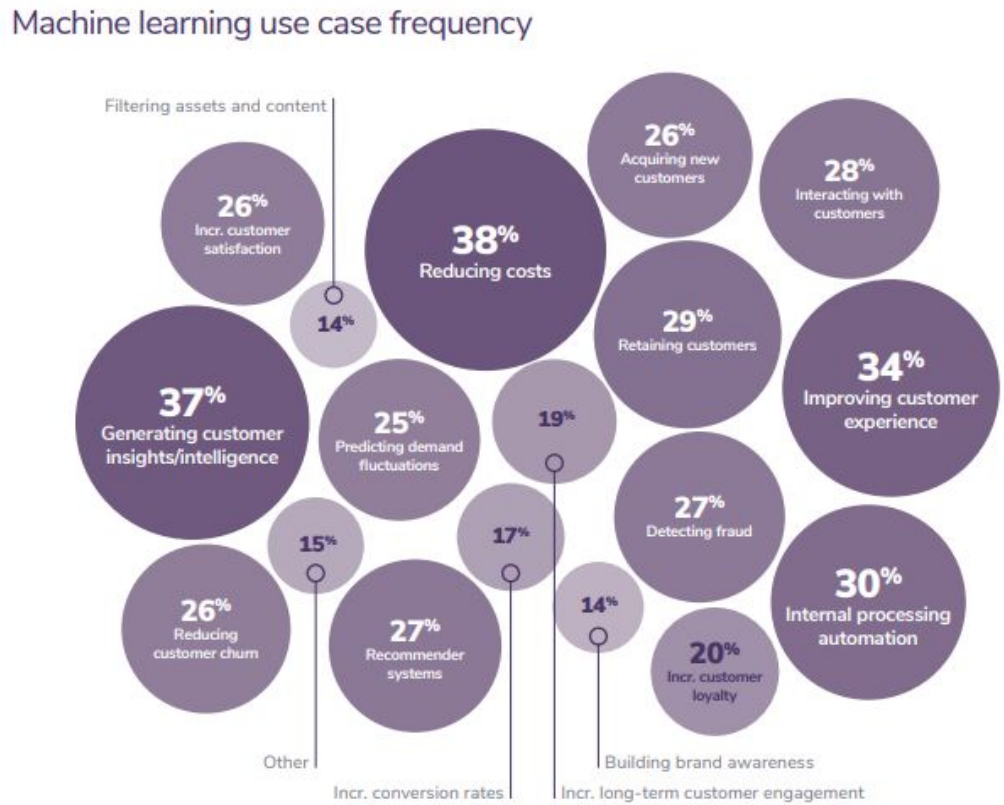

Modelado
Elección del Baseline y entrenamiento de los modelos
Análisis de error

Proceso iterativo

Crear un modelo de machine learning es un proceso iterativo



Problemas comunes de ML



https://www.coriniumintelligence.com/2020-state-of-enterprise-machine-learning-algorithmia-whitepaper-download

Seis recomendaciones para elegir el modelo

1. Evita la trampa de los modelos que estado del arte
2. Empieza por los modelos más simples
3. Evita el bias en la selección de modelos
4. Evalua la buena performance ahora vs. la buena performance después
5. Evalúa los trade-offs
6. Entiende los supuestos del modelo

Entrenar nuestros propios modelos o usar pre-entrenados?

- Depende el problema que querramos resolver, tecnología con la que contemos, recursos técnicos, datos, tiempos y costos
- Hay muchos problemas que ya se encuentran solucionado por modelos pre-entrenados de diferente costo y disponibilidad por ejemplo:
  - Modelos de voz a texto y texto a voz (speech-to-text o text-to-speech)
  - Modelos de Lenguaje - LLMs - ChatGPT, Gemini, Gemma, Claude, Llama
  - Modelos de generación de imágenes
  - Modelos de detección de objetos en imágenes (Object Detection)
  - Modelos de detección de caras
- Estos modelos pueden ser sistemas completos con una librería o un archivo de pesos para realizar la predicción
- También, hay mucho código para entrenar nuestros propios modelos en base a los datos con diferentes tecnologías. Un ejemplo clásico puede ser los modelos de detección de fraude o recomendadores

Model Cards - Entiende los supuestos de los modelos

- Todos los modelos que usemos, debemos conocer la teoría que hay detrás. Hay muchos supuestos no escritos que acompañan al tipo de modelo e implementación. Utilizar el modelo desde una API es sencillo pero nuestro trabajo es conocer el detalle para garantizar un buen sistema de ML
- Las diferentes empresas suelen ofrecer mucha documentación en torno al modelo. Una concepto que está ganando fuerza es el de **Model Cards.**
- También es una buena práctica para nuestros modelos.
- 4 ejemplos:
  - MedLM Model Card
  - GPT 4o System Card
  - Gemma 2 Model Card
  - Imagen 3 Model Card in Paper
- Datos que suelen incluir: descripción del modelo, descripción de los datos y el entrenamiento, riesgos, consideraciones éticas de AI, metodologia de evaluación. Algunos incluso incluyen el impacto en la sociedad y evaluaciones hechas por 3rd parties.
- Ya se haciendo nuestro modelo o usando un modelo pre-entrenado es interesante contar con estas cartas de modelo. También es interesante incluirlas en los proyecto de ML que tengamos de la facultad, el laburatorio o la empresa.

Model Cards

No se trata sólo de un modelo

- Crear sistemas de Machine Learning no se trata sólo de aplicar una técnica en particular sino que hay una **combinación de técnicas y prácticas propias del dominio**
- Ejemplo: los modelos del lenguaje además de utilizar una red neuronal del modelo transformers para su entrenamiento, incluyen mejoras que tienen que ver con el aprendizaje por refuerzos con feedback humano en el loop (RLHF). Además, constituyen un sistema con varios filtros y revisiones para no proveer textos incorrectos o dañinos.
- Ejemplo 2: Un sistema de recomendación tiene particularidades propias del dominio del problem. Los sistemas de recomendación se clasifican en item-item o user-item. Despues pueden ser sistemas content-based o collaborative-filtering.
- Por lo cual, muchas veces se habla genéricamente de modelo o sistema de ML pero que incluye muchas otras técnicas y modelos enlazados

Técnicas Clásicas para modelos del Machine Learning

- Regresión Lineal
- Regresión Logística
- Arboles de Decisión
- Random Forest
- Support Vector Machines
- XGBoost
- K-Nearest Neighbors
- Neural Networks
  - Convolutional Neuronal Networks
  - Recurrent Neuronal Networks - LSTM - GRU
  - Fully Connected Layers networks
  - Generative Adversarial Networks
  - Autoencoders
  - Transformers
- Técnicas propias del dominio: Q-Learning (Aprendizaje por refuerzo)

muchas más!!

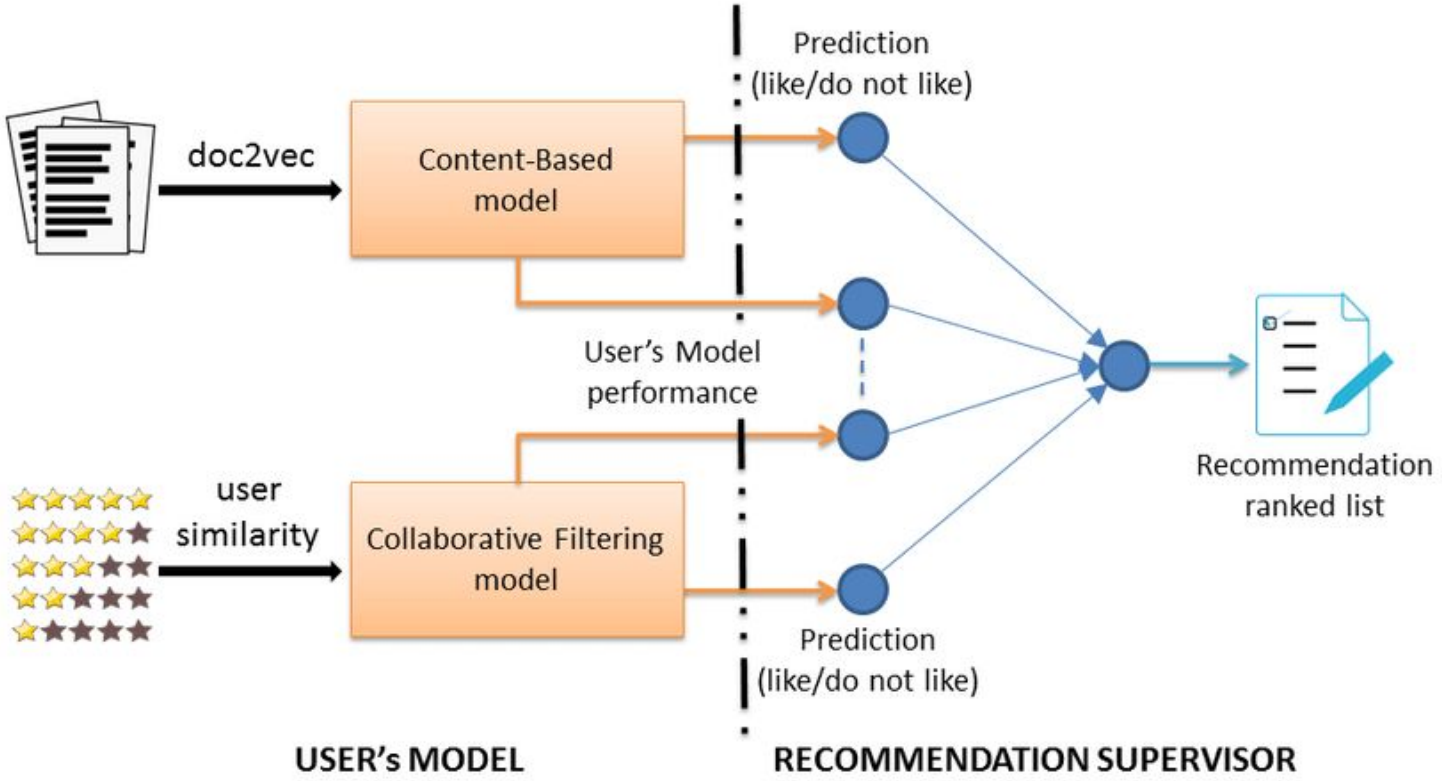
Importante! Nuestro objetivo es resolver un problema
Primero el problema y luego buscamos la técnica acorde al problema.
La técnica siempre está definida o creada en los papers alrededor de un set de datos y problema a resolver.
Un mismo problema puede ser resuelto por varias técnicas con distintos tradeoff

Recomendaciones para elegir el modelo

¿Qué recomendaciones le darían a alguien que quiere crear un modelo de ML?

¿Cómo hacen uds para elegir el modelo una vez que tienen los datos y el problema solucionar definido?

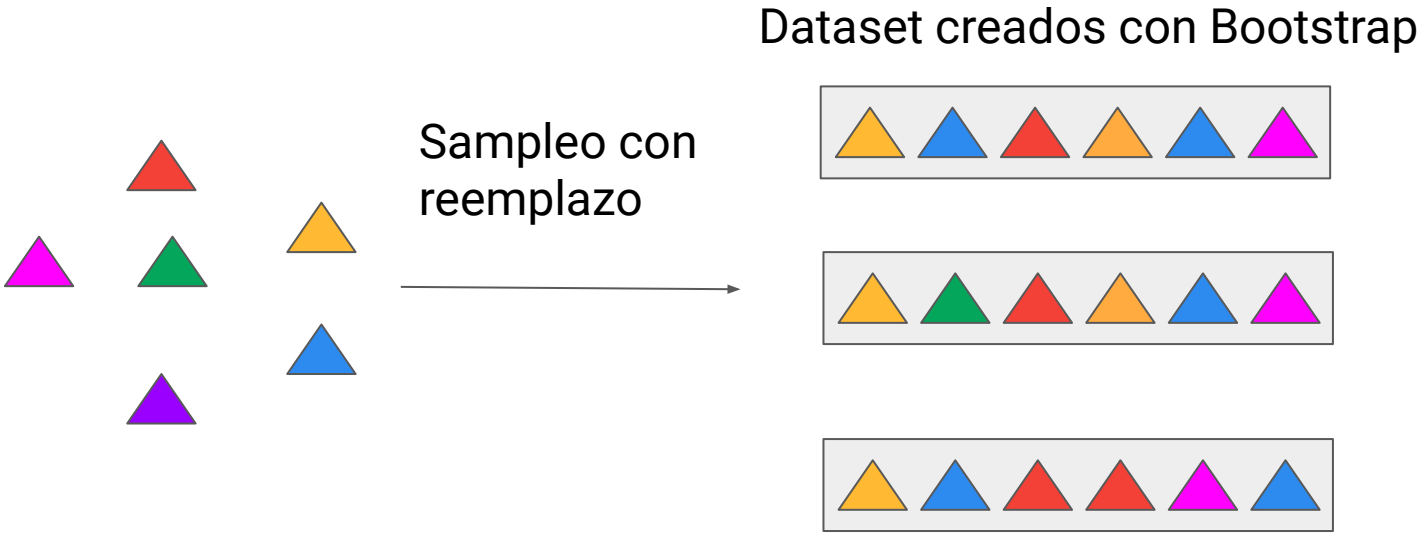
Modelos híbridos - Ensembles



Towards a Deep Learning model for Hybrid Recommendation

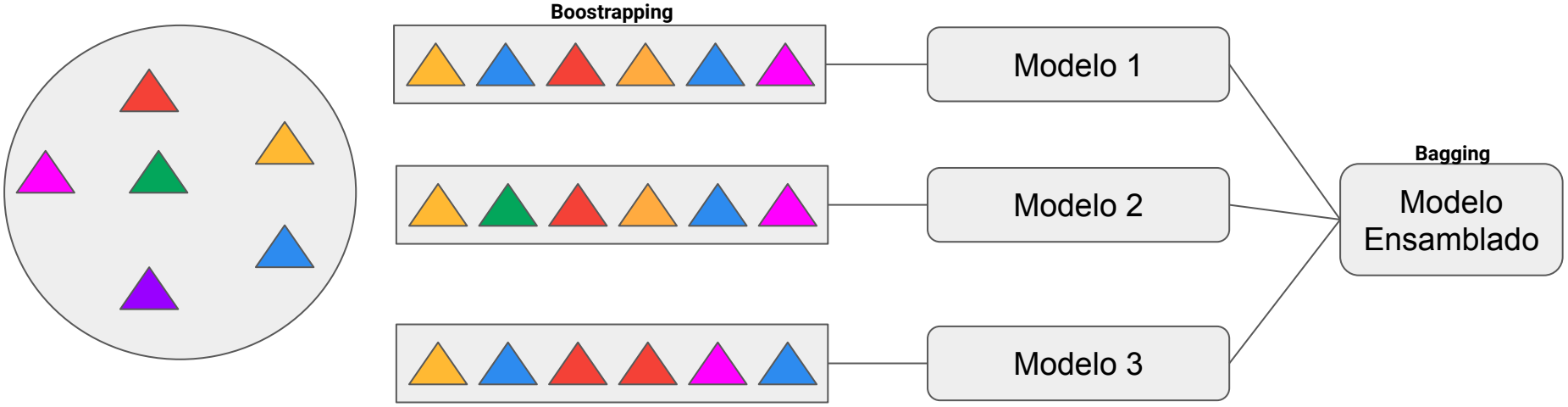
Bootstrap

- Samplear con reeemplazo de un dataset.
- Es una técnica muy interesante cuando se cuentan con pocos datos.
- Se usa en el algoritmo Random Forest



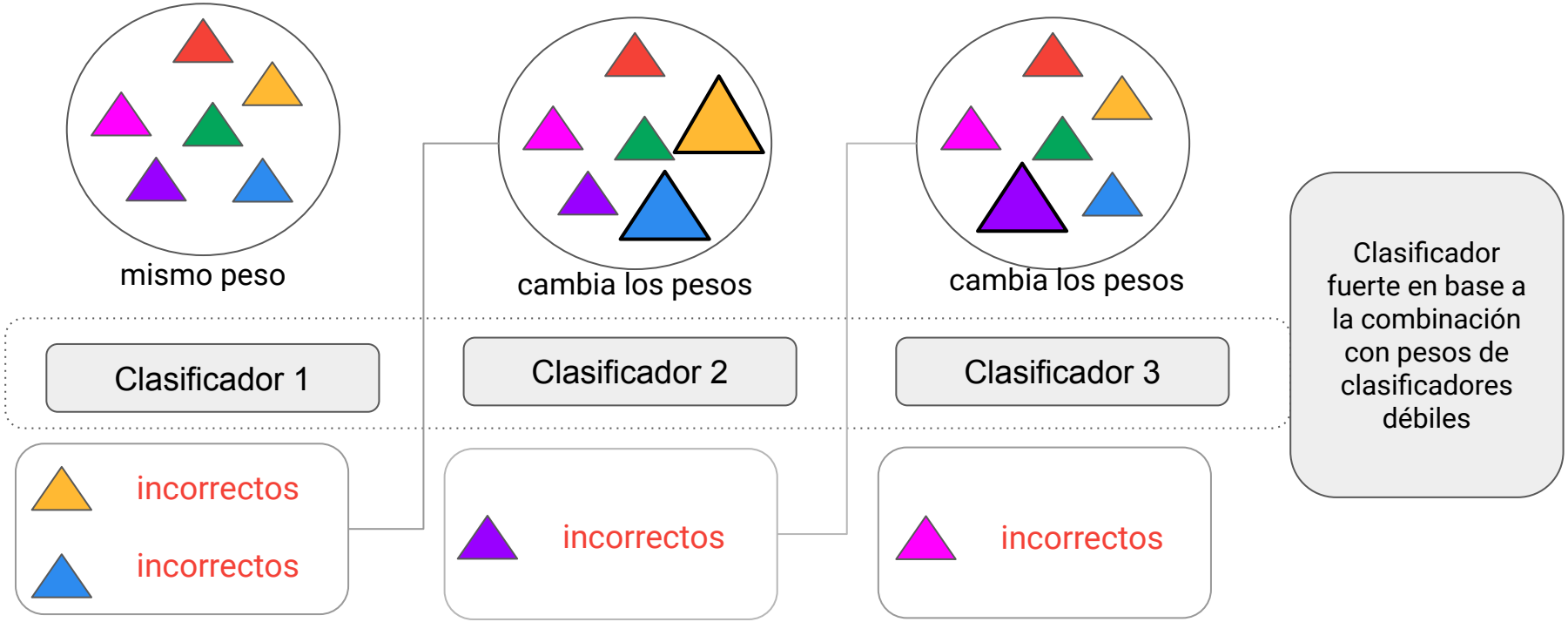
Modelo Ensamble: Bagging - Bootstrap aggregating

- Reducen la varianza. Evitan el overfitting.
- Usa bootstrapping
- Se entrenan varios modelos usando un dataset creado con bootstrapping sobre el dataset original. Luego, se obtiene un modelo ensamblado.
- Al momento de predecir, se decide por voto mayoritario de los submodelos (en caso de clasificación) o el promedio (en caso de regresión)



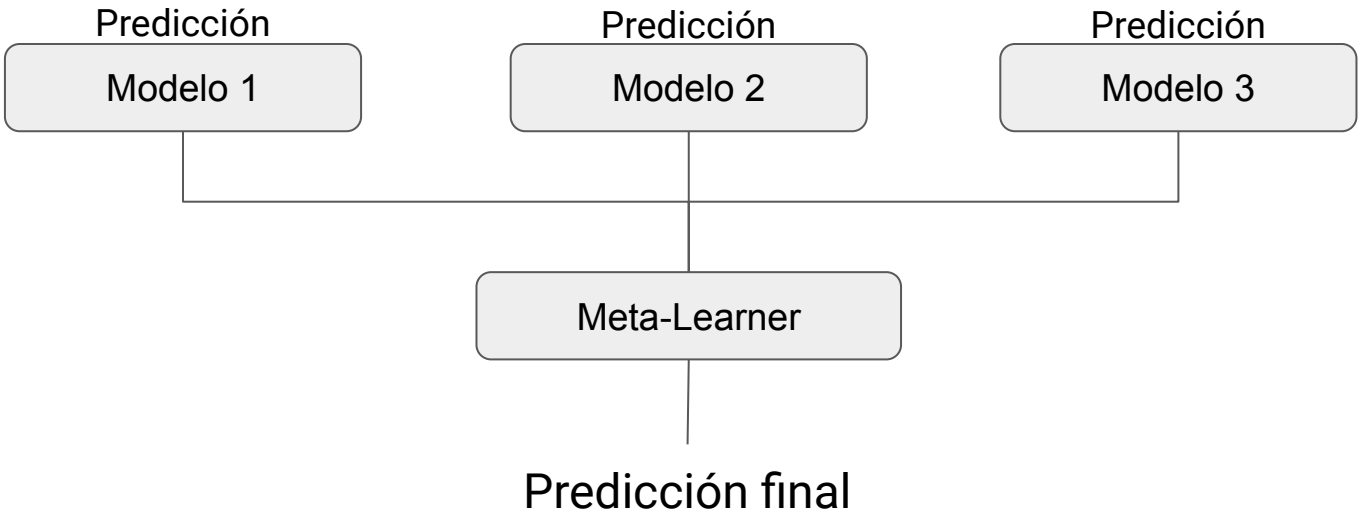
Boosting

- Se entrena varias veces asignando pesos a los elementos mal clasificados, por lo cual se dice que transforma weak learners en strong.
- Ejemplo, Gradiente Boosting Machine (GBM), XGBoost



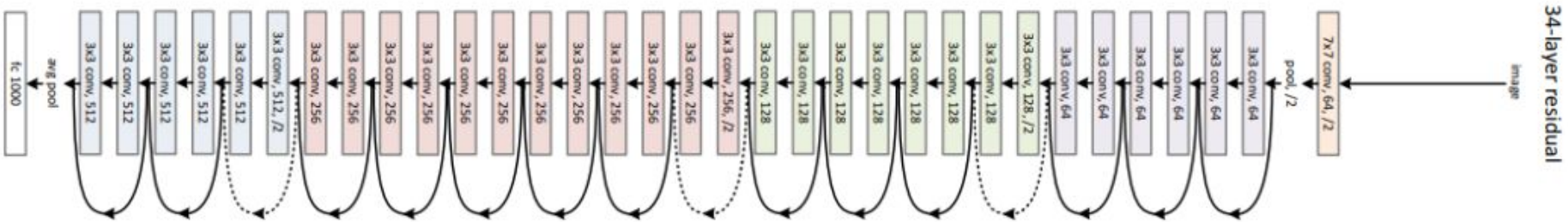
Stacking

- Entrenar varios modelos (learners) y crear un meta-learner que combinalas salidas de los base-learners como predicción final.
- Una combinación posible es por voto para clasificación o promedio para regresión

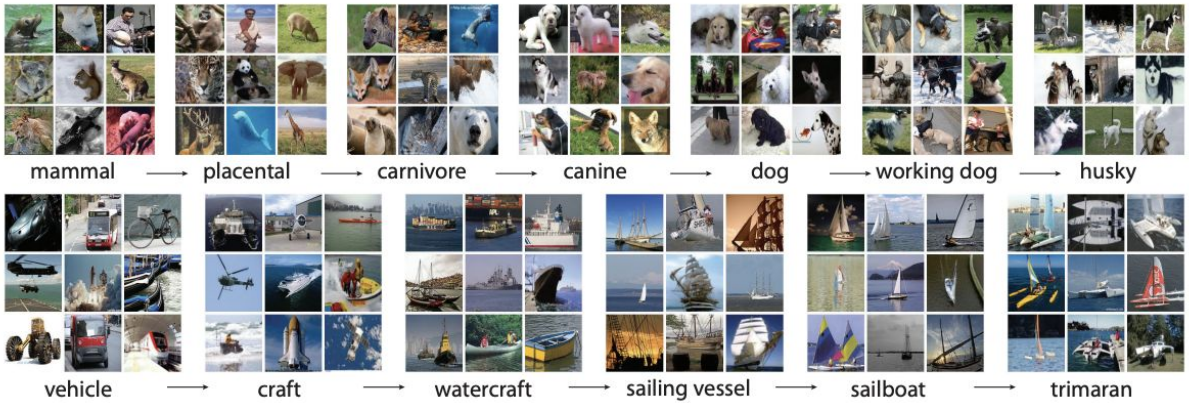


Transfer Learning - Ejemplo red pre-entreanda

ResNet 34 layers



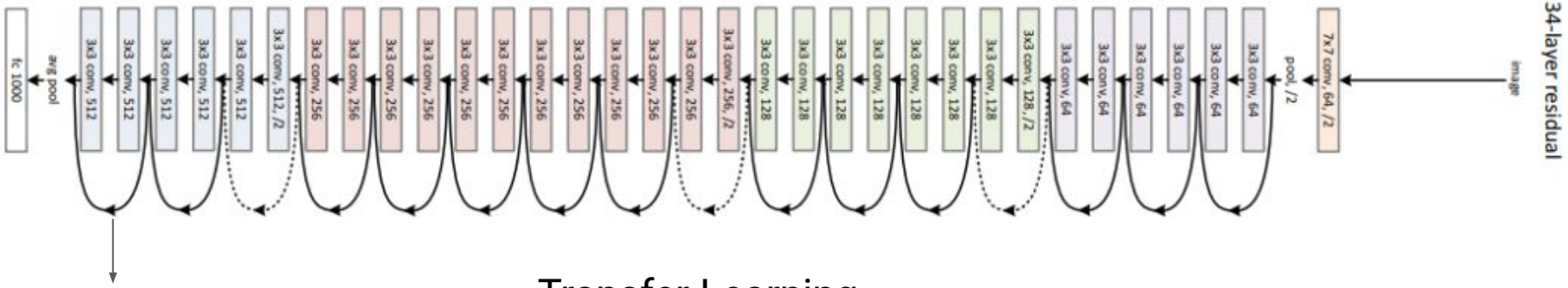
ImageNet



https://arxiv.org/abs/1512.03385

Transfer Learning - Ejemplo red pre-entreanda

ResNet 34 layers - entrenada sobre ImageNet clasifica imágenes en 1000 categorías



Transfer Learning

Matriz de Pesos w

fc 2

Re-entrenamos una red más pequeña usando los pesos de Resnet (el aprendizaje) y los datos nuevos. Obtenemos un clasificador de 2 clases para el dataset nuevo.

https://arxiv.org/abs/1512.03385

AutoML

- Forma automatizada de buscar un modelo de Machine Learning
  - Soft AutoML - Hyperparameters Tuning
    - Automatizar la búsqueda de los mejores hiperparámetros
      - Grid search
      - Random search
      - Bayesian Optimization
    - Framework: auto-sklearn, Keras Turner
  - Hard AutoML - Architecture search and learned optimizer
    - Automatizar no sólo la búsqueda del modleo sino también la construcción del mismo.
    - Area de investigación conocida como architectural search, or neural architecture search (NAS) for neural networks

https://www.automl.org/book/ Editors: By Frank Hutter, Lars Kotthoff and Joaquin Vanschoren
Efficientnet https://research.google/blog/efficientnet-improving-accuracy-and-efficiency-through-automl-and-model-scaling/

Hardware para el entrenamiento del modelo

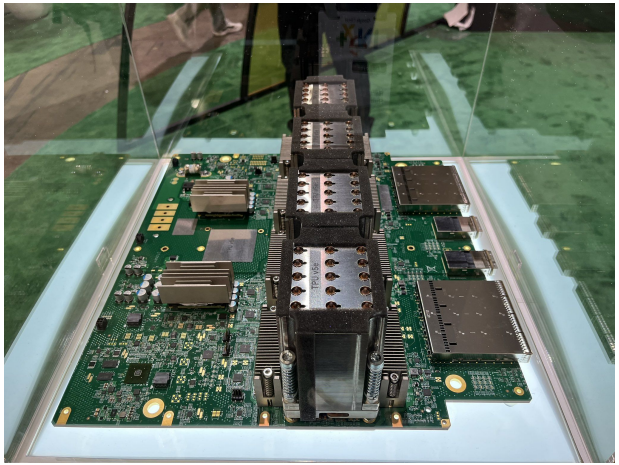
Para el entrenamiento de los modelos hay dos consideraciones de Hardware fundamentales:

- Memoria: algunos algoritmos mantienen los datos en memoria durante el procesamiento. Grandes volumenes de datos en memoria puede ser un limitante para el entrenamiento de grandes modelos.
- Procesamiento: los modelos grandes, por ejemplo, redes neuronales sobre imágenes precisan mucho procesamiento para hacer cálculos matriciales. Para mejorar la velocidad es conveniente entrenar estos modelos en GPU o TPU en lugar de CPU

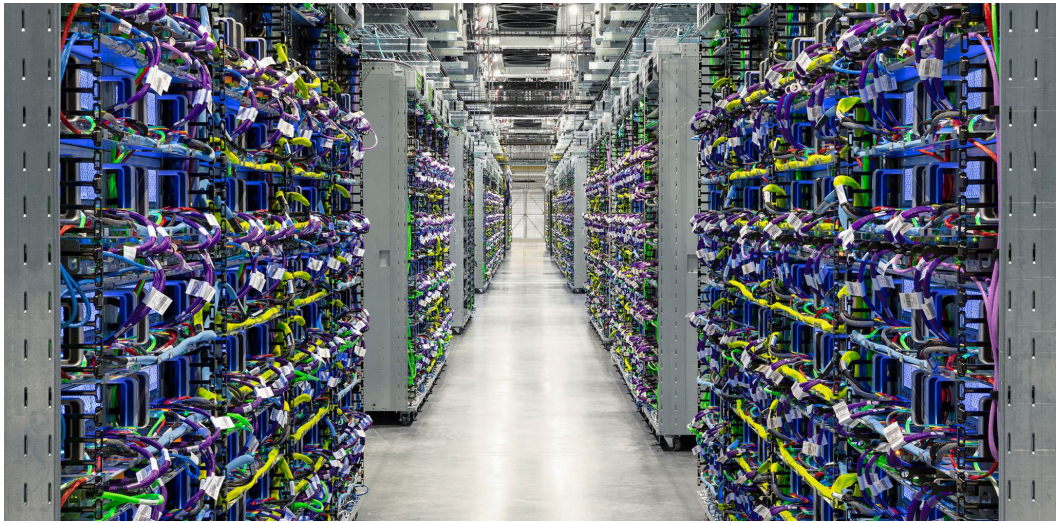
Existen alternativas como el entrenamiento distribuido para mejorar los tiempos de respuesta y procesar modelos muy profundos. Igual estas dos consideraciones funcionan como limitante en el tiempo de entrenamiento y factibilidad.

TPU - Tensor Processing Unit

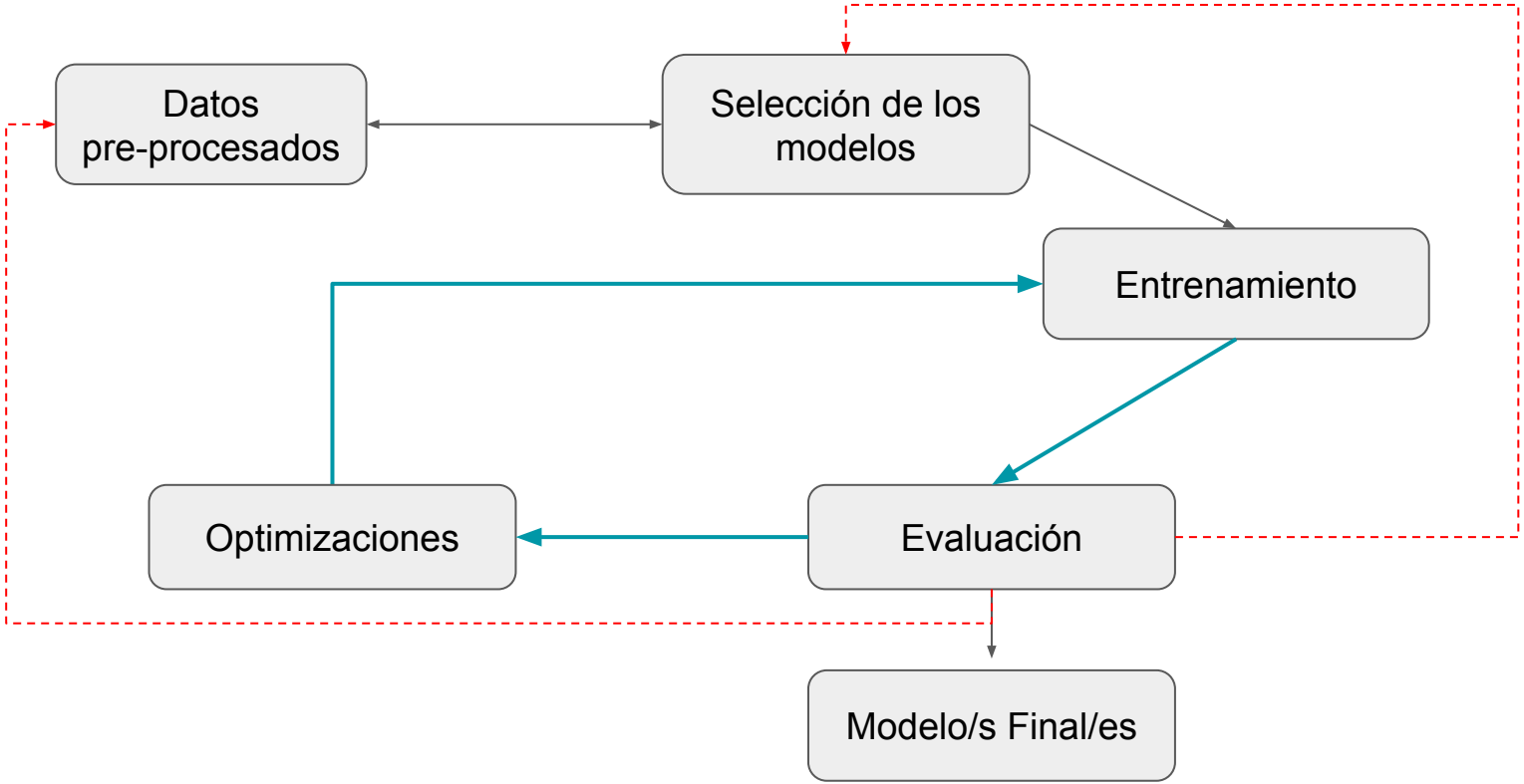
- Es procesador especial usado para Machine Learning y que está disponible en la nube de Google
- Los procesadores gráficos GPU fueron originalmente diseñadas para el procesamiento de gráficos y tienen una mayor precisión en punto flotante que la TPU
- Dado que para un modelo de ML el exceso de precisión no es diferencial, se creó la TPU cuya velocidad es mayor al GPU y está especialmente diseñada para problemas de ML.
- Tensorflow permite correr modelos en CPU, GPU o TPU



TPU v5e



Desarrollo de un modelo ML



Parámetros e Hiperparámetros

- **Parámetros**: se aprenden durante el entrenamiento desde los datos por ejemplo, matriz de pesos y bias en redes neuronales o los coeficientes en la regresión lineal $F(x) = w^T x + b$
- **Hiperparámetros**: son las configuraciones para el entrenamiento del modelo por ejemplo: learning rate, epochs, batch size, capas en la red neuronal, arboles en un random forest
- Técnicas para buscar los Hiperparámetros:
  - Grid Search
  - Random Search
  - Bayesian Search
- Encontrar los hiperparámetros que obtengan las mejores métricas es una tarea manual o puede ser automatizada mediante un algorimo o herramientas (optuna, hyperopt)

Hyperparameter
Learning rate
Learning rate schedule
Optimizer choice
Other optimizer params (e.g., Adam beta1)
Batch size
Weight initialization
Loss function
Model depth
Layer size
Layer params (e.g., kernel size)

Métricas y Funciones de pérdida

- Las métricas a seleccionar para nuestro modelo dependen del problema, algoritmo seleccionado y lo que querramos lograr.
- Estas métricas nos permiten determinar que tan bueno es el modelo.
- Las métricas más populares son:
  - Accuracy
  - Precision
  - Recall
  - F1-Score
  - Confusion Matrix
  - AUC ROC
  - AUC PR
  - BLEU Score
- Las funciones de pérdidas más polulares son:
  - Cross-entropy Loss
  - Mean Squared Error (MSE)
  - Intersection over Union (IoU)
  - Mean Absolute Error (MAE)
  - Root Mean Squared Error (RMSE)

Actividad: Exploremos las métricas y funciones de pérdida más populares con sus problemas relacionados

Leamos juntos el paper “Loss functions and Metrics in Deep Learning”.

1. Lean de la introducción el punto 1.1. Loss Functions vs Performance Metrics
2. Elijan del índice algún área de problemas que les interese explorar las métricas:

- Regression
- Classification
- Computer Vision Tasks
- Natural Language processing
- Retrieval Augmented Generation (RAG)

Y exploren las diferentes métricas y funciones de pérdidas del área. ¿Cuáles conocen? ¿Cuáles son nuevas para uds? Intenten comprender algunas de las expresiones matemáticas y contexto de uso de las que más les interese.
Noten que el paper es un pre-print.

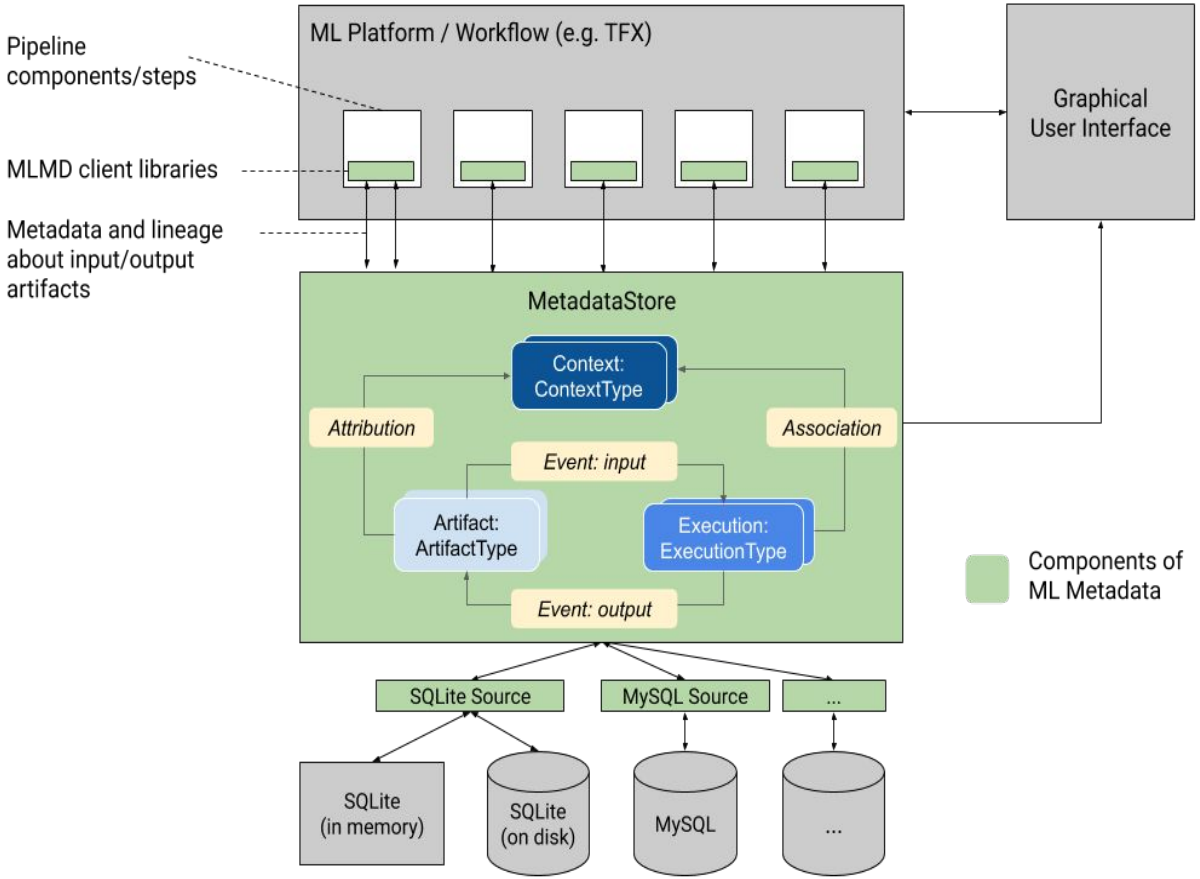
Metadatos en el entrenamiento de modelos

- **Experiment Tracking**: el proceso de anotar el progreso y resultado de los distintos experimentos.
- Se puede trackear:
  - curvas de pérdida (loos)
  - métricas (f1 score, accuracy, precision, recall, perplexity)
  - velocidad de entrenamiento, procesamiento
  - uso de memoria, CPU, GPU
  - parametors, hiperparámetors, evolución del learning rate por ejemplo
  - data, predicción, y label esperado

TFX - Tensorflow Extended - ML Metadata

ML Metadata es un biblioteca para registrar y obtener los datos respecto a los flujos de trabajo. En cada ejecución podemos registrar y conocer:

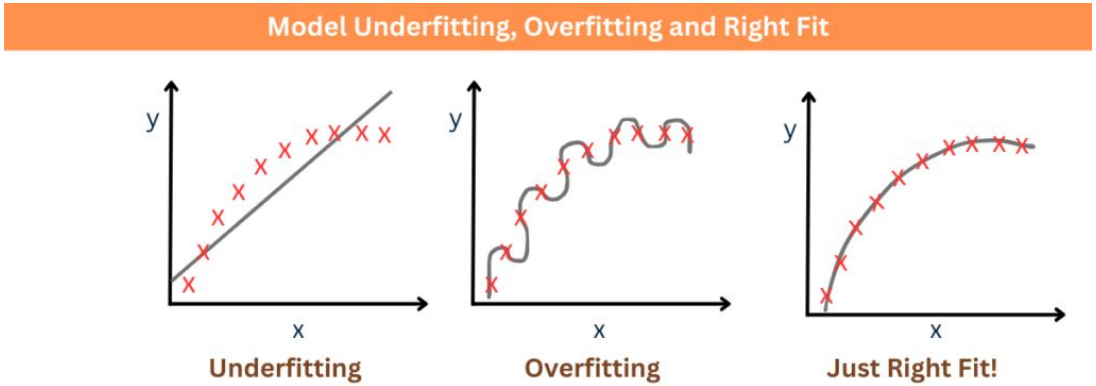
- ¿En qué conjunto de datos se entrenó el modelo?
- ¿Cuáles fueron los hiperparámetros utilizados para entrenar el modelo?
- ¿Qué tramo de tubería creó el modelo?
- ¿Qué entrenamiento condujo a este modelo?
- ¿Qué versión de TensorFlow creó este modelo?
- ¿Cuándo se impulsó el modelo fallido?



Versionado de Código y Datos

- Versionado de código puede hacerse con Git.
- También es importante conocer con que datos se entrenó el modelo. Este versionado es complejo por la cantidad de datos.
- Las herramientas de versionado nos ptemrient revertir cambios, pero es imposible hacerlo con el dataset y a veces, lleva a duplicar datos.
- Versionado y reproducibilidad son dos principios del MLOps que tanto el versionado de código, datos y de experimentos nos permite garantizar.
- Es interesante poder definir un proceso de versionado y trackeo de datos y experimentación para todo el laboratorio/área. Considerar que los modelos tienen que sobrevivir a las personas / equipos y proveer certezas sobre su funcionamiento.

Overfitting, Underfitting



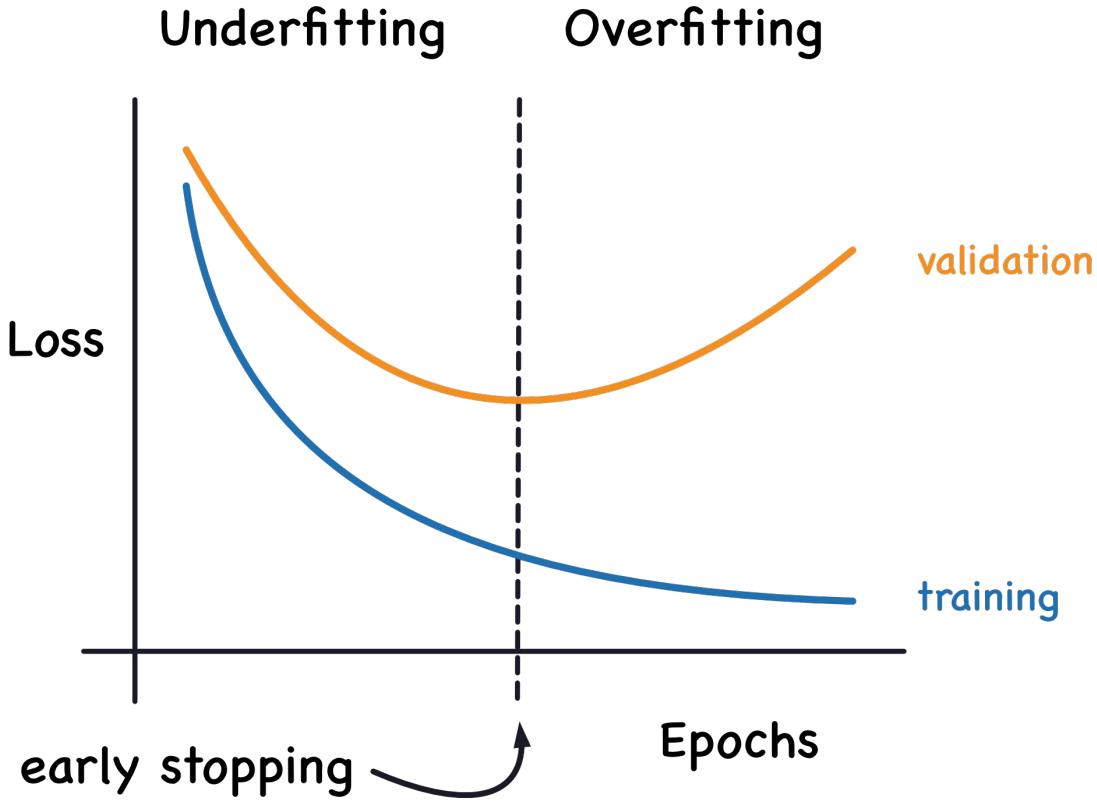
Bias	High	Low	Low
Variance	Low	High	Low

En overfitting, el modelo esta entrenado ajustandose mucho a los datos de entrenamiento que le hes imposible generalizar, es decir, predecir bien sobre nueva data no vista durante entrenamiento

Técnicas para evitar overfitting:

- usar métodos de regularización como dropout, L1 o L2 regularization
- agregar más data
- entrenar menos épocas o usar early stoping
- usar batch normalization
- data augmentation
- cross-validation
- reducir la cantidad de capas de la red
- mejorar la ingenieria de features: selección de features, manejar los valores nulos, normalizar, sacar outliers.

Overfitting, Underfitting y Early Stopping



Baselines

- Es necesario tener un modelo contra el cual evaluar. Depende del caso de uso y el contexto de negocio o de la investigación.
- Ideas para baselines:
  - Baseline aleatorio: supongamos que los datos estan desbalanceados y se hacen predicciones aleatorias cual sería el accuracy y F1. Eso puede ser un primer baseline.
  - Heurística: se puede hacer alguna heuristica simple que realice la predicción y tratar de mejorarla
  - Zero rule baseline: es una heurística en el cual siempre predice correctamente la clase más probable. Sistema de recomendación, los más vendidos.
  - Humano: por ejemplo, un algoritmo de AI que juegue videojuegos y compararlo con la performance de los mejores jugadores.
  - Existing solutions: Muchas veces, este modelo puede ser algún modelo desarrollado actualmente y que se encuentra productivo.

Ejemplo de uso de Baselines y métricas en la literatura

	Gemini Ultra (pixel only)	Gemini Pro (pixel only)	Gemini Nano 2 (pixel only)	Gemini Nano 1 (pixel only)	GPT-4V	Prior SOTA
MMMU (val) Multi-discipline college-level problems (Yue et al., 2023)	59.4% pass@1 62.4% Maj1@32	47.9%	32.6%	26.3%	56.8%	56.8% GPT-4V, 0-shot
TextVQA (val) Text reading on natural images (Singh et al., 2019)	82.3%	74.6%	65.9%	62.5%	78.0%	79.5% Google PaLI-3, fine-tuned
DocVQA (test) Document understanding (Mathew et al., 2021)	90.9%	88.1%	74.3%	72.2%	88.4% (pixel only)	88.4% GPT-4V, 0-shot
ChartQA (test) Chart understanding (Masry et al., 2022)	80.8%	74.1%	51.9%	53.6%	78.5% (4-shot CoT)	79.3% Google DePlot, 1-shot PoT (Liu et al., 2023)
InfographicVQA (test) Infographic understanding (Mathew et al., 2022)	80.3%	75.2%	54.5%	51.1%	75.1% (pixel only)	75.1% GPT-4V, 0-shot
MathVista (testmini) Mathematical reasoning (Lu et al., 2023)	53.0%	45.2%	30.6%	27.3%	49.9%	49.9% GPT-4V, 0-shot
AI2D (test) Science diagrams (Kembhavi et al., 2016)	79.5%	73.9%	51.0%	37.9%	78.2%	81.4% Google PaLI-X, fine-tuned
VQAv2 (test-dev) Natural image understanding (Goyal et al., 2017)	77.8%	71.2%	67.5%	62.7%	77.2%	86.1% Google PaLI-X, fine-tuned

Table 7 | **Image understanding** Gemini Ultra consistently outperforms existing approaches even in zero-shot, especially for OCR-related image understanding tasks for natural images, text, documents, and figures without using any external OCR engine ('pixel only'). Many existing approaches fine-tune on the respective tasks, highlighted in gray, which makes the comparison with 0-shot not apples-to-apples.

SOTA - State of the Art
Mejor modelo es Baseline

[Gemini: A Family of Highly Capable Multimodal Models]

Evaluación

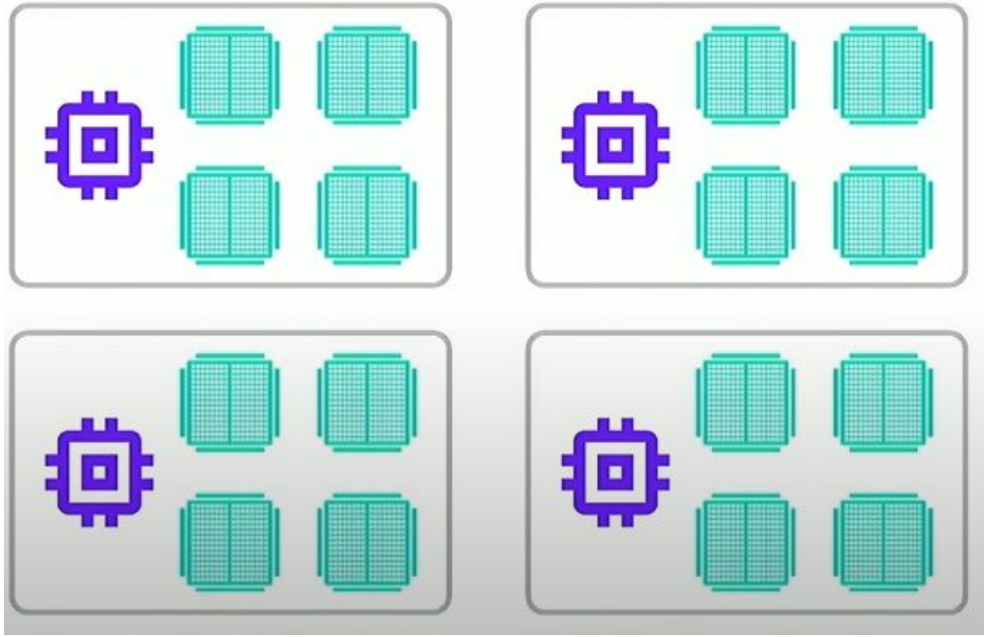
- En investigación, es suficiente con usar métricas pero en la práctica, el modelo productivo es más complejo y se pueden usar otras estrategias para garantizar que el modelo es robutos, justo, calibrado y tenga sentido para el caso de uso.
- Evaluaciones adicionales:
  - Tests con perturbaciones: hacer pequeños cambios en el dataset de testeo que afecten al modelo para ver que tan bien se desempeñaría en esos casos
  - Tests de invariancia - Bias - AI Responsable: mantener los inputs como estan pero cambiar información sensible.
  - Tests con resultados experados dirigidos: algunos cambios en las entradas deben producir determinadas salidas, probarlo.
  - Model calibration: la probabilidad marginal esperada para cada modelo de cada clase se debe mantener y cumplir, sino el modelo no esta calibrado. Hay formas de detectarlo con sklearn.
  - Confianza: hacer varias predicciones para detectar casos que no produzcan confianza o errores muy graves, predicciones sin sentido
  - Evaluaciones por corte de datos: separar los datos y medir las performance en cada subset.

Entrenamiento Distribudo

- Existen formas que nos permiten entrenar modelos en forma paralela.
- Util para:
  - optimizar el uso de recursos de cómputo
  - solucionar problemas de falta de memoria en modelos grandes (por ejemplo, redes neuronales profundas con muchos parámetros)
- Por ejemplo, **Tensorflow** trae distintas modalidades en tf.distribute usadas para paralelismo en datos:
  - MirroredStrategy
  - MultiWorkerMirroredStrategy
  - TPUStrategy
  - ParameterServerStrategy
  - CentralStorageStrategy
- También, se puede hacer paralelismo de modelos entrenando las capas de la red neuronal en distintas máquinas

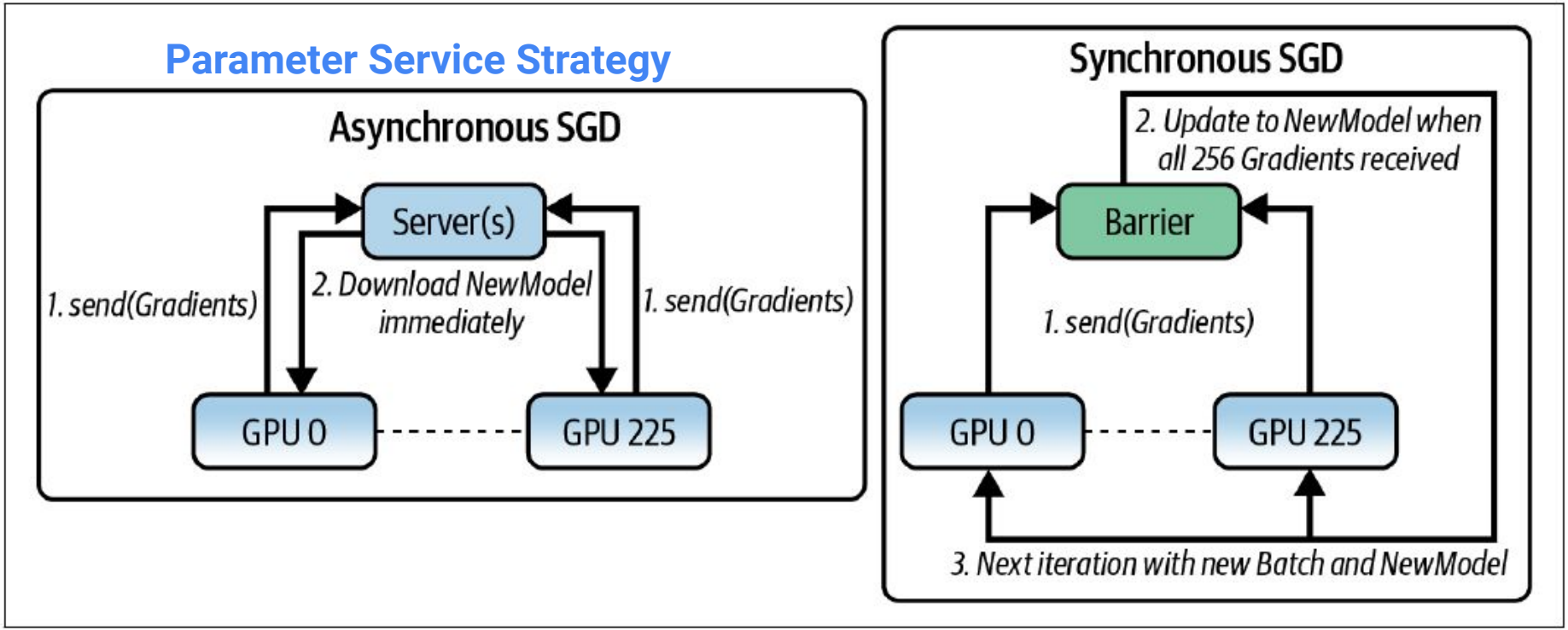
Entrenamiento Distribudo - MultiworkerMirroredStrategy

- Multiple workers con multiples CPU/GPU
- Sincronizados
- Tienen un paso lock donde sincronizan los gradientes a cada paso



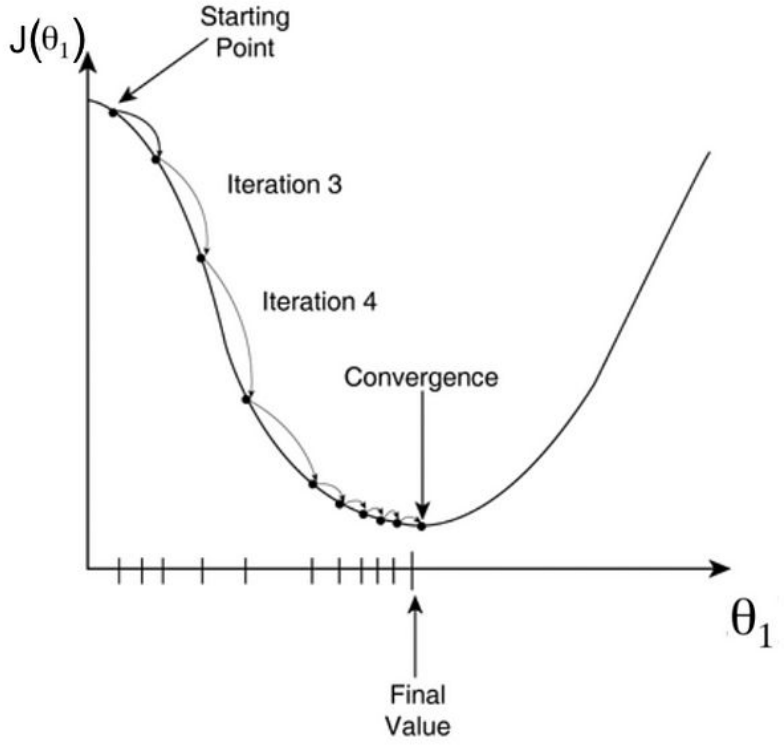
Entrenamiento Distribudo - Paralelismo de Datos

MirroredStrategy
MultiWorkerMirroredStrategy
TPUStrategy



Chip Huyen, Designing Machine Learning System, 2022

SGD - Stochastic Gradient Descent



Cost Function – "One Half Mean Squared Error":

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Objective:

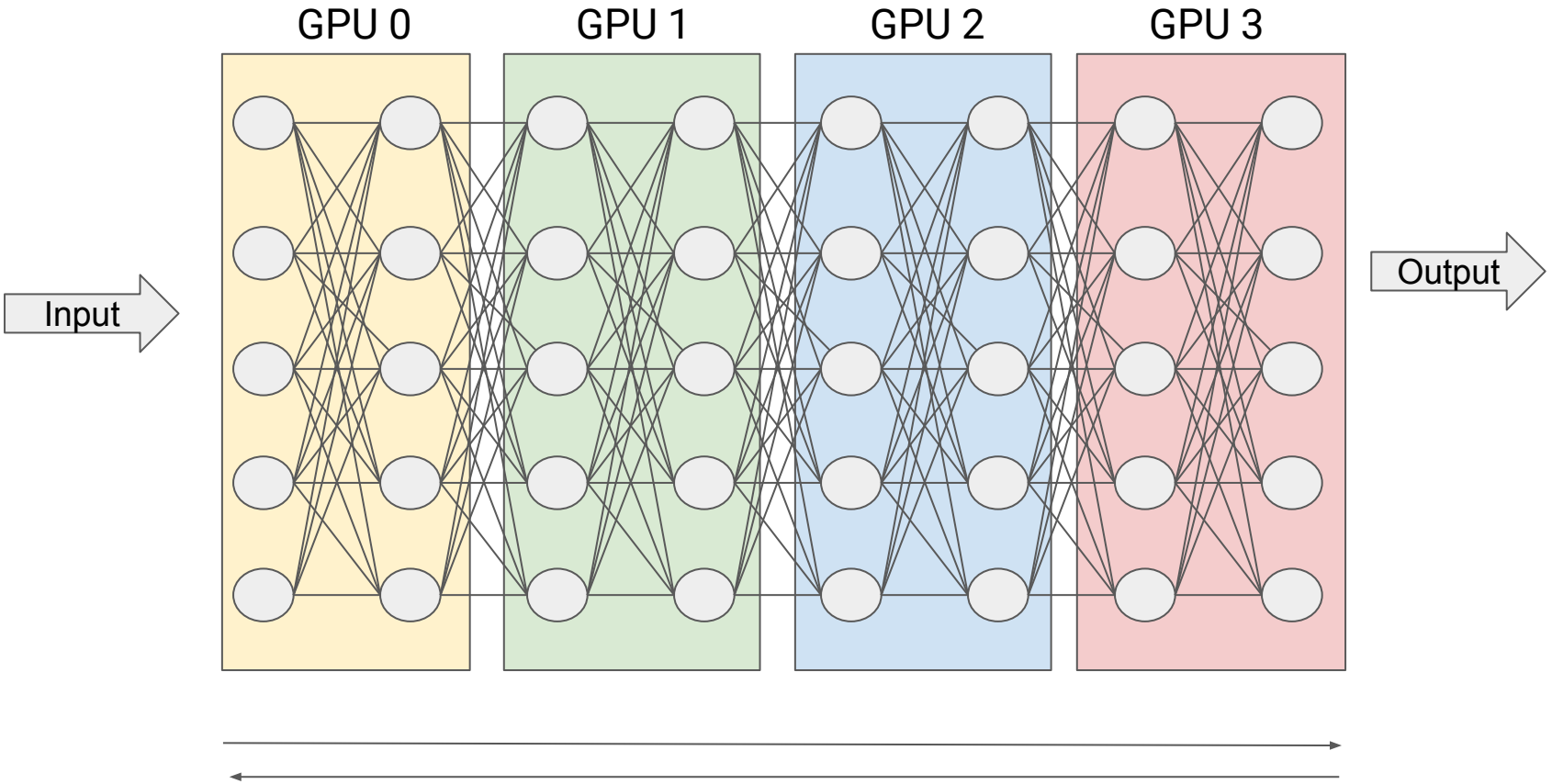
$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

Derivatives:

$$\frac{\partial}{\partial\theta_0}J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$\frac{\partial}{\partial\theta_1}J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)\cdot x^{(i)}$$

Entrenamiento Distribuido - Paralelismo de Modelo



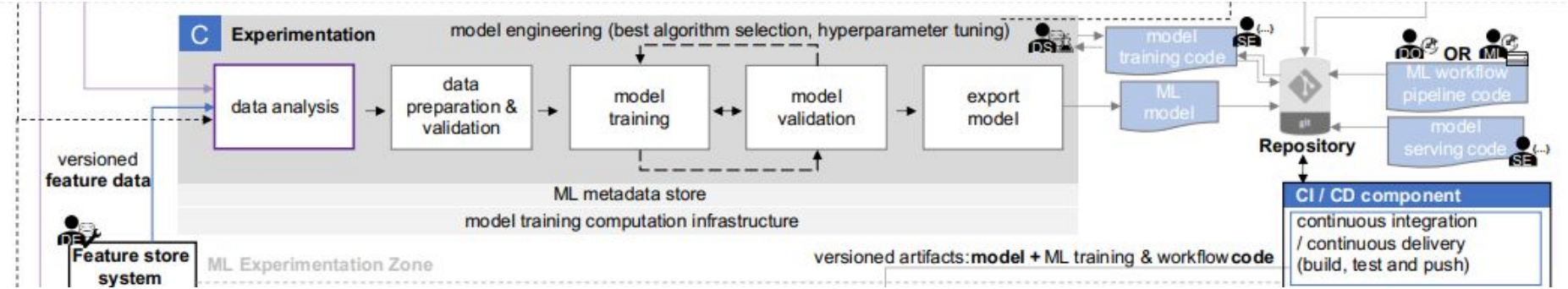
Empaquetar el modelo

- La mayor parte de las liberías nos permite descargar los datos necesarios para la predicción del modelo en algún archivo serializado o con formato que luego podemos usar al momento de predicción
- Por ejemplo,

```
from sklearn.linear_model import LogisticRegression
import pickle
model = LogisticRegression()
model.fit(X_train, Y_train)
# save the model to disk
pickle.dump(model, open('model.pkl', 'wb'))
```

```
# import tensorflow dependencies
from tensorflow.keras.models import Sequential, model_from_json
from tensorflow.keras.layers import Dense

# define model architecture
model = Sequential()
model.add(Dense(12, input_dim=4, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
model.fit(X_train, y_train, epochs=150, batch_size=10, verbose=0)

# save model and its architecture
model.save('model.h5')
```

Training Pipelines - Continuous Training



Referencias

https://www.tensorflow.org/guide/distributed_training?hl=es-419
https://www.youtube.com/watch?v=yH1cF7GnoIo&t=1862s
https://www.tensorflow.org/guide/keras/distributed_training
EfficientNet Paper
Chip Huyen, Designing Machine Learning Systems, chapter 6, Model Development and Offline Evaluation