Escuela Técnica Superior de Ingeniería Informática

Máster Universitario en Ingeniería Informática

Tema 4.3
Entrenamiento y Despliegue

Deep Learning

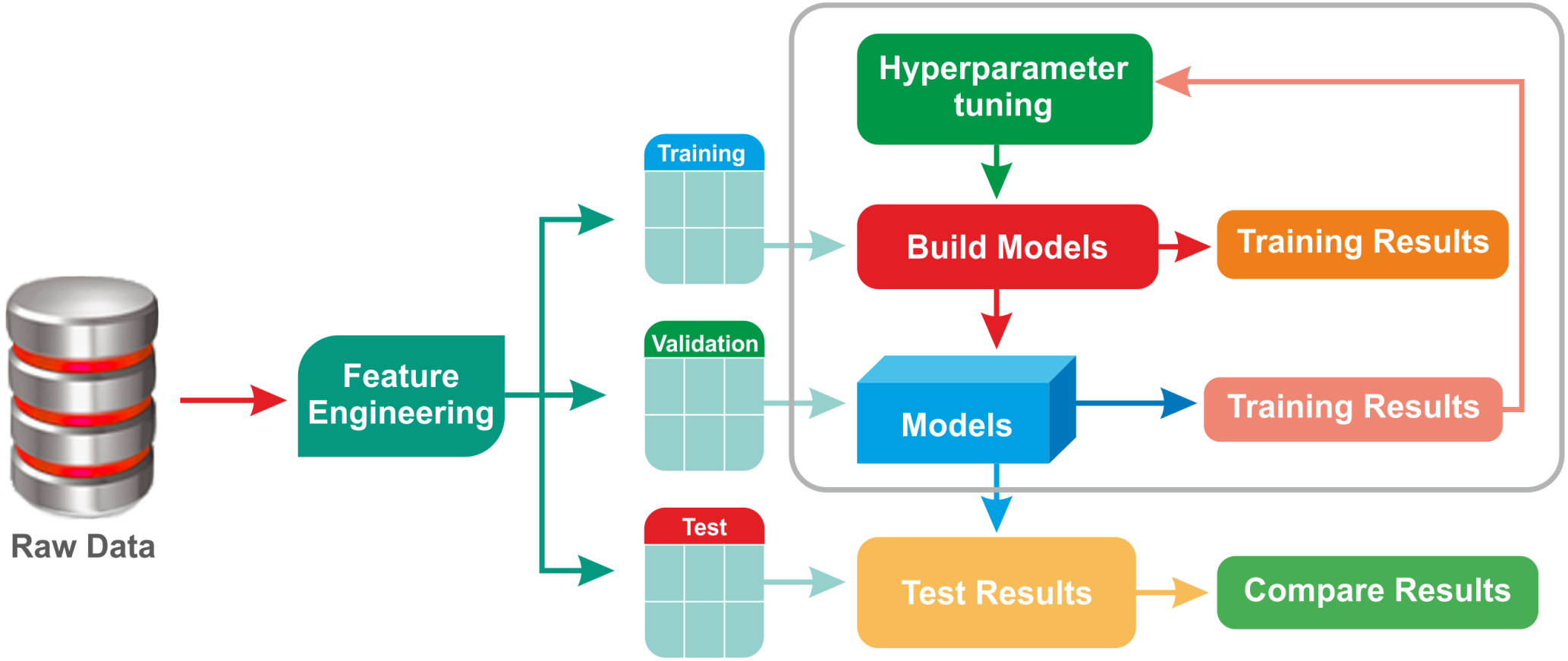
Máster Oficial en Ingeniería Informática

Universidad de Sevilla

Contenido

- El proceso de aprendizaje
- Optimización de hiperparámetros
- Métodos de despliegue eficiente

El proceso de aprendizaje



El proceso de aprendizaje

1. **Primera configuración**
   - *Preprocesamiento de datos, aumentado de datos*
   - *Elección de una arquitectura, funciones de activación, inicialización de pesos*
   - *Elección de método de regularización*
   - *Elección de función de coste y método de optimización*
2. **Dinámica de entrenamiento**
   - *Monitorización del entrenamiento (lo trabajaremos en las prácticas)*
   - *Optimización de parámetros*
   - *Optimización de hiperparámetros (NAS, AutoML)*
3. **Evaluación**
   - *Ensamblado de modelos*

Optimización de hiperparámetros

Model Design

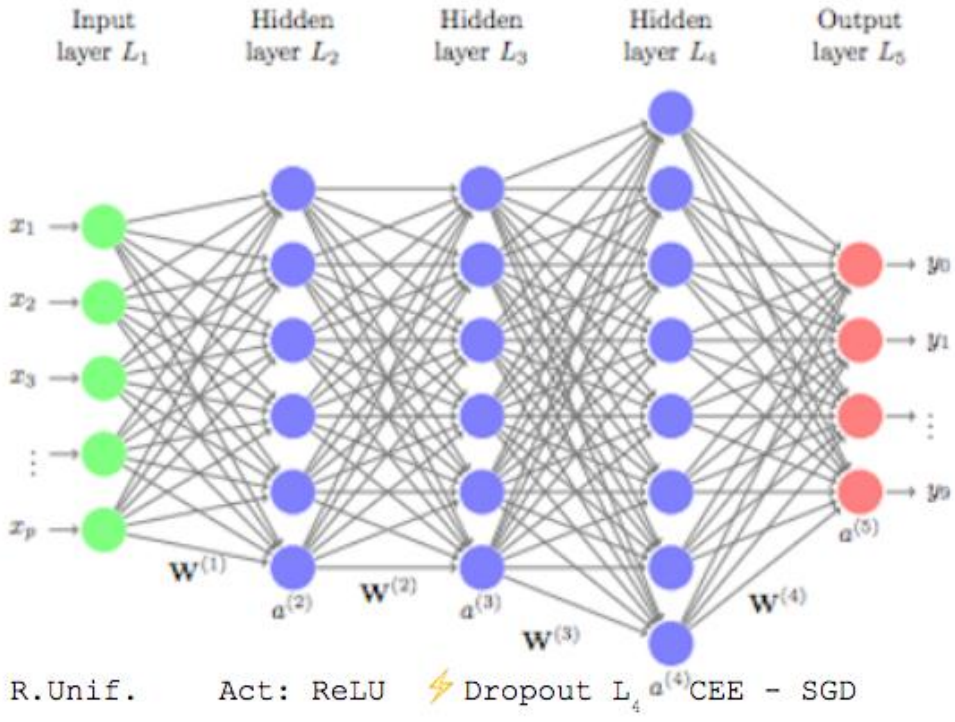
- Weight init.: Random Uniform
- Act.: ReLU
- Loss: CEE
- # Hidden Layers: 3
- # Units per layer {p, p+1, p+1, p+3, 10}
- Optimizer: SGD
- Dropout layer: L_4

Hyperparameters

- Learning rate
- Dropout Rate
- Batch size

Model Parameters

- $W^{(1)} => W^{(4)}$

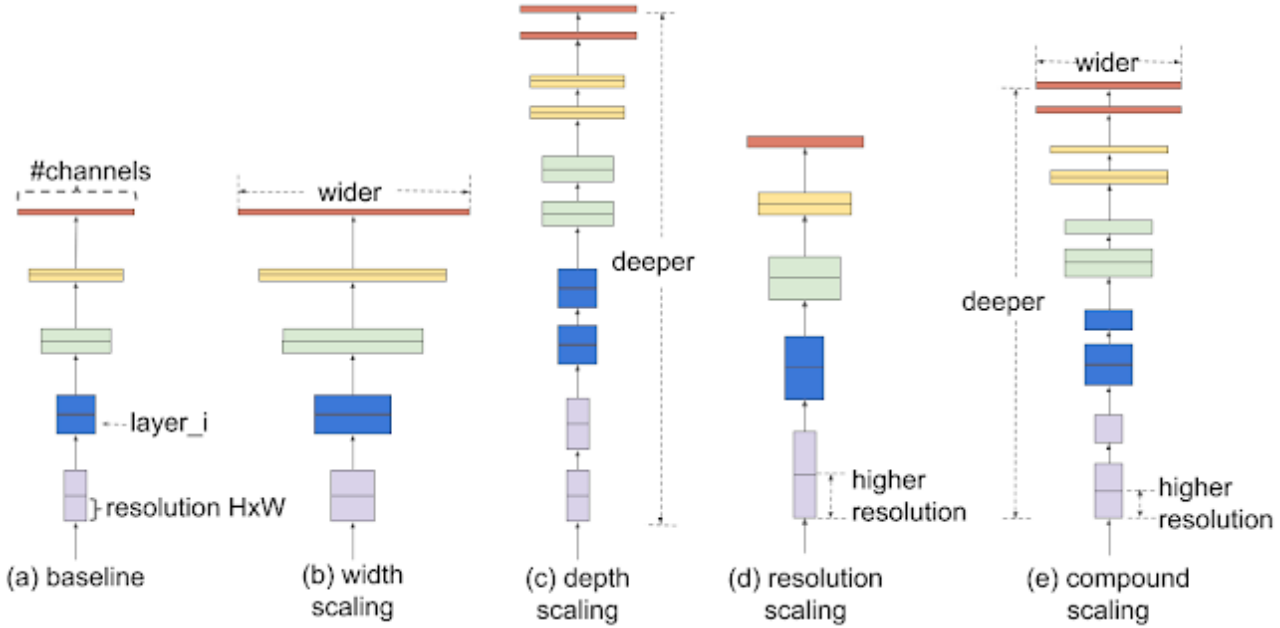


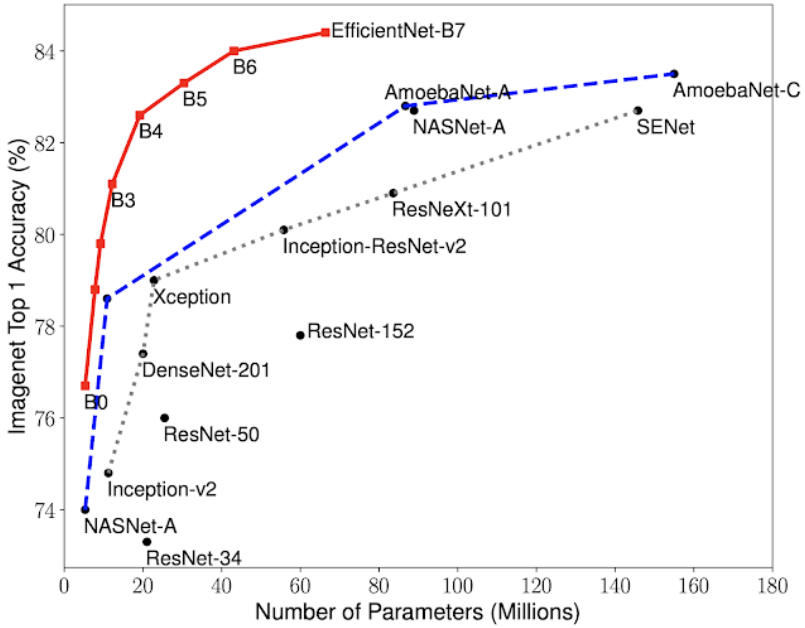
Optimización de hiperparámetros

- **Neural Architecture Search (NAS)** o **AutoML**
- **Hiperparámetros** con los que jugar:
  - La **arquitectura** de la red
  - El **learning** rate, su decaimiento y tipo de actualización
  - El **momentum** (si se escoge este método con SGD)
  - **Regularización** (L2/Dropout)
  - Tamaño del **batch**
- Métodos:
  - Manual: Babysitting o “prueba y error”
  - Búsqueda paralela: Grid Search, Random Search, Bayesian Optimization, Reinforcement Learning, Búsqueda Evolutiva

Optimización de hiperparámetros: Ejemplo

- **EfficientNet (2019)**: Combinación de escalas según anchura, profundidad y resolución. Redes obtenidas mediante AutoML





Optimización de hiperparámetros: babysitting

- Comenzar **observando la pérdida** obtenida del modelo:
  - Al comienzo, desactivar la regularización
  - Observar que la pérdida incrementa al introducir regularización
  - Comprobar que podemos hacer sobreajuste sobre unos cuantos datos
- Comenzar con un **factor pequeño de regularización** y **encontrar el learning rate** que decrementa la pérdida:
  - Si la pérdida no decrece: learning rate muy bajo
  - Si la pérdida explota: learning rate demasiado alto
  - Probar valores entre 10^{-3} y 10^{-5}.

Optimización de hiperparámetros: babysitting

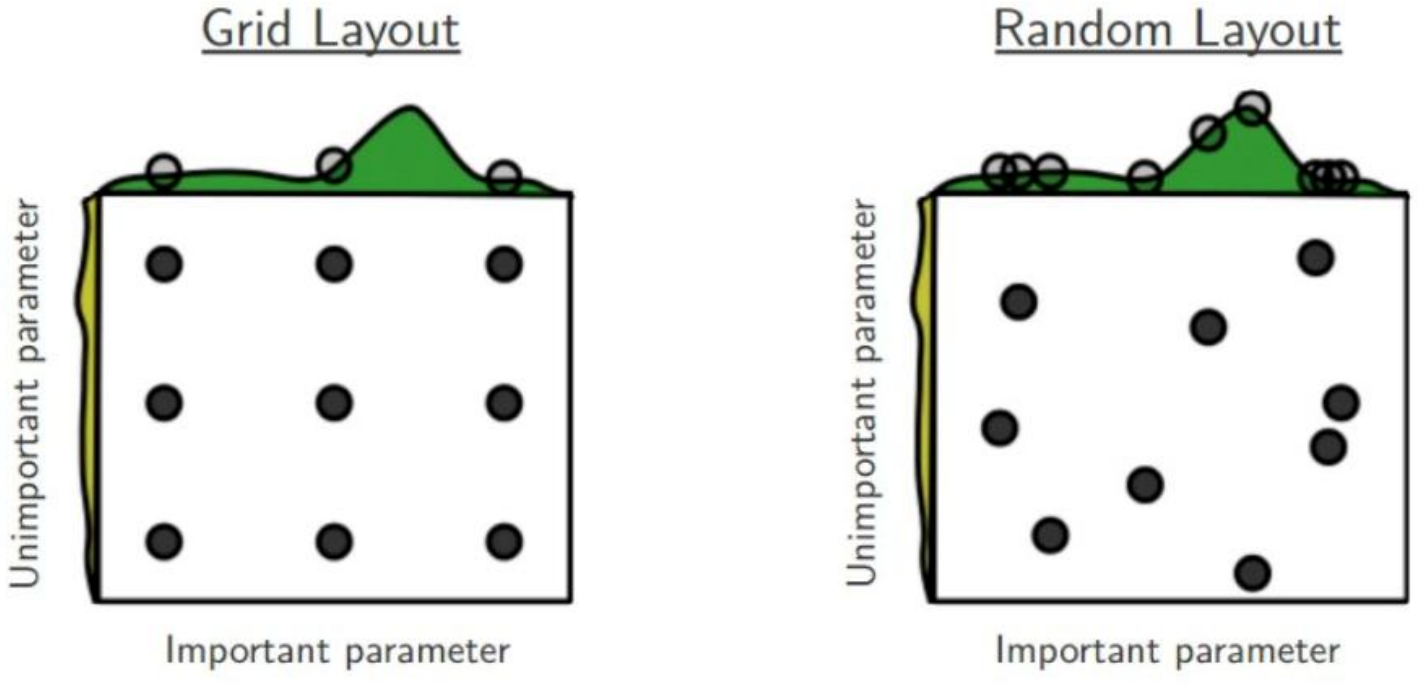
- Hacer validación cruzada:
  - Primero con pocas épocas para ver rápidamente los valores que funcionan bien.
  - Probar valores en escala logarítmica! Es decir: lr= 10^{-3}, 10^{-4}, 10^{-5}
  - Después de elegir
    - Probar más épocas.
    - Afinar los valores de los hiperparámetros (esta vez no logarítmicamente)

Optimización de hiperparámetros: Grid search

- Probar configuraciones en paralelo mediante un grid:
  - Definir un grid de n dimensiones, donde cada dimensión es un hiperparámetro.
  - Para cada dimensión, definir un rango de posibles valores. Por ejemplo, batch size = 4, 6, 8, 16, 32, 64, 128; learning rate = 0,01; 0,001; 0,0001
  - Busca para todas las posibles configuraciones y espera los resultados para encontrar la mejor combinación.

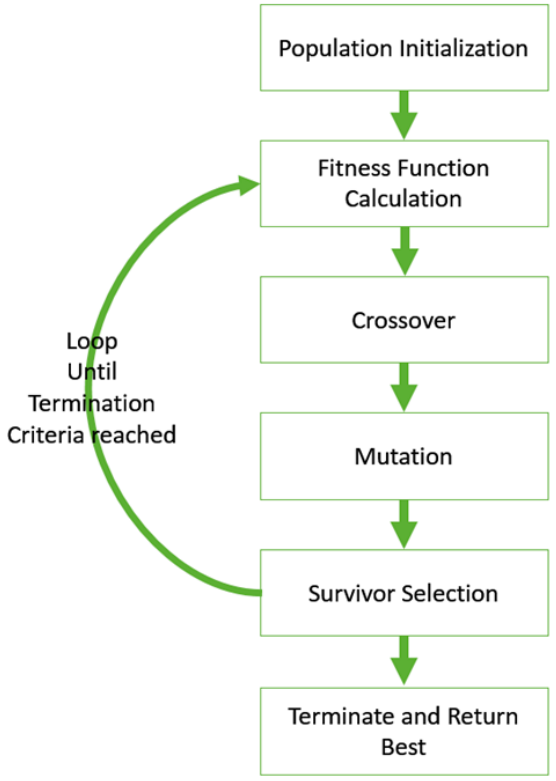
Optimización de hiperparámetros: random search

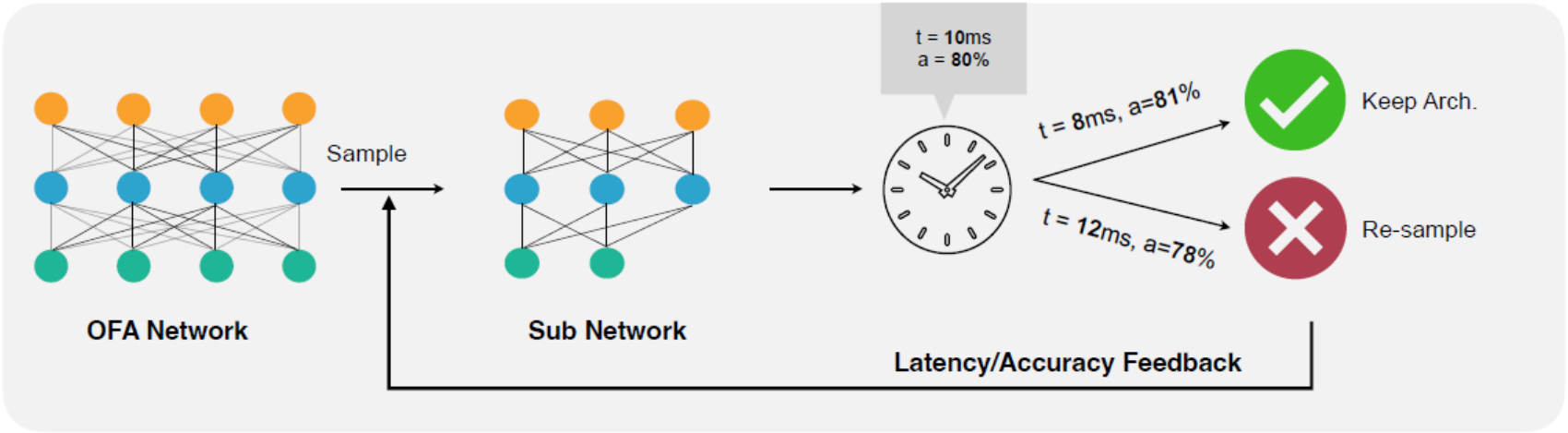
- Introducido en [Bergstra y Begio, 2012]:
- La diferencia real está en el primer paso, random search escoge los puntos aleatoriamente del espacio de configuración



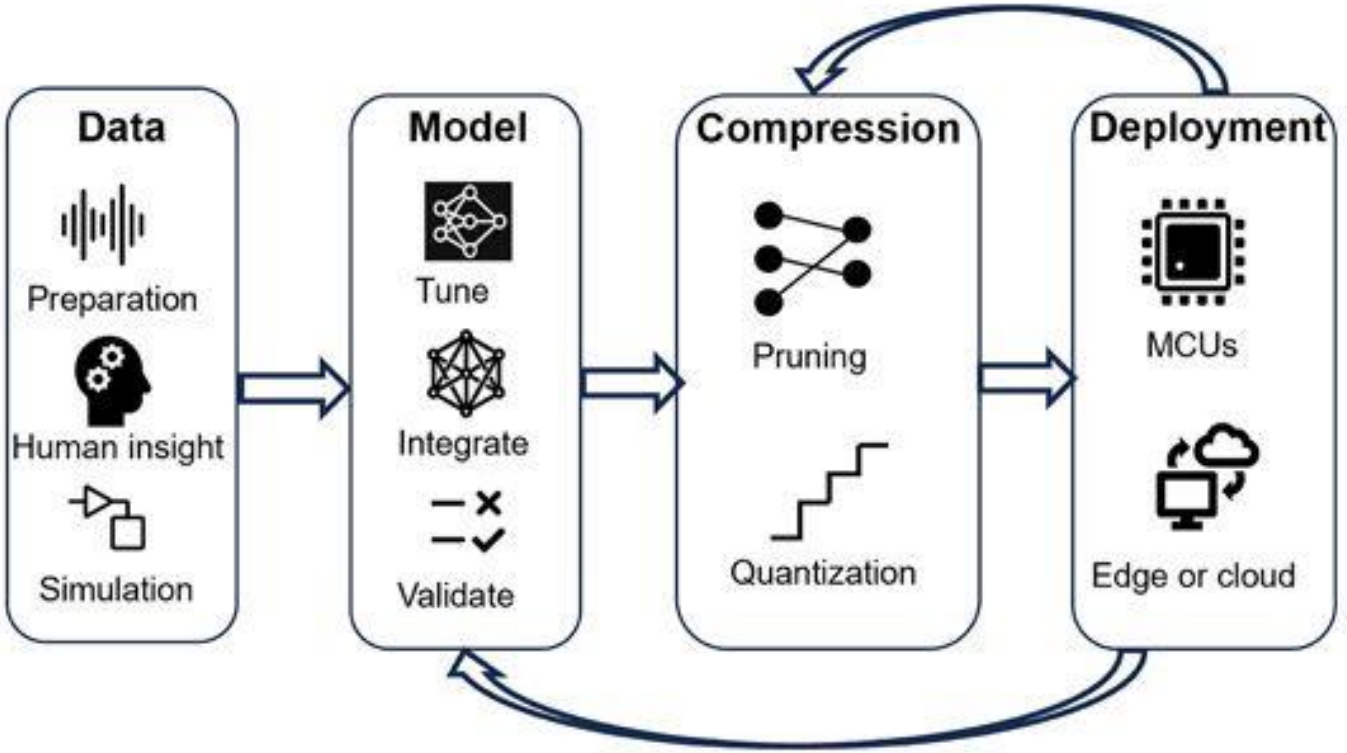
Optimización de hiperparámetros: Ejemplo

- **Búsqueda evolutiva**





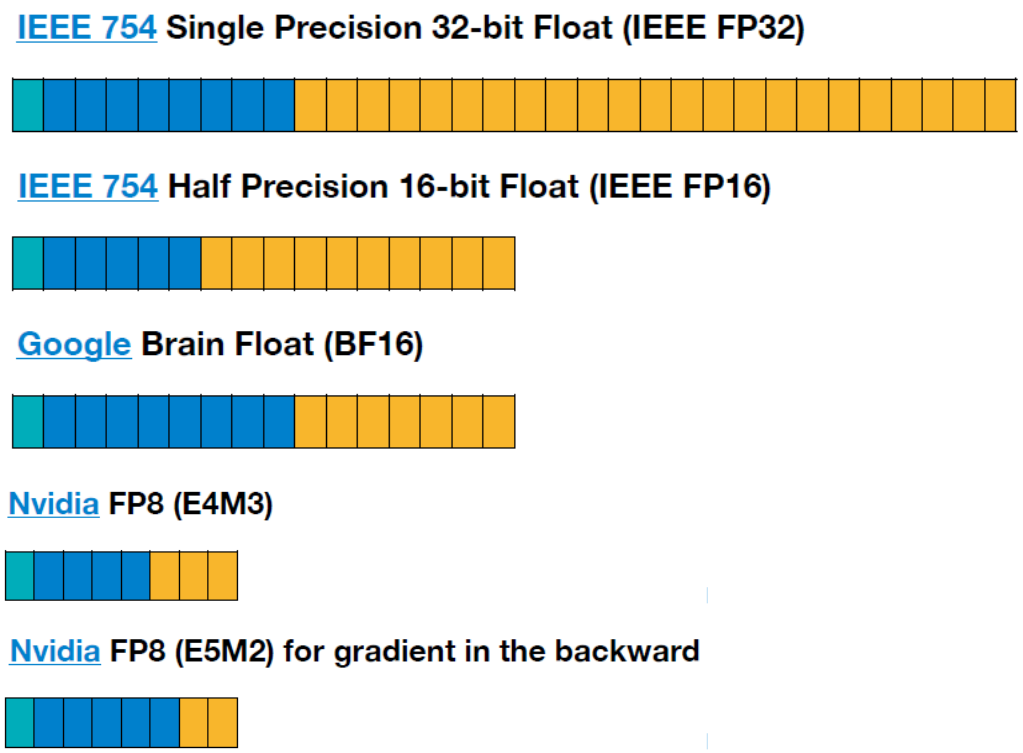
Métodos de Despliegue Eficiente



https://www.mathworks.com/company/technical-articles/rapid-deployment-of-deep-learning-on-edge-devices.html

Métodos de Despliegue Eficiente

- **Cuantización de pesos:**
  - Uso de representación con menos bits
  - Menos memoria
  - Si hardware adaptado, más cálculos en paralelo



INT4

S			
0	0	0	1
0	1	1	1

FP4 (E1M2)

S	E	M	M
0	0	0	1
0	1	1	1

FP4 (E2M1)

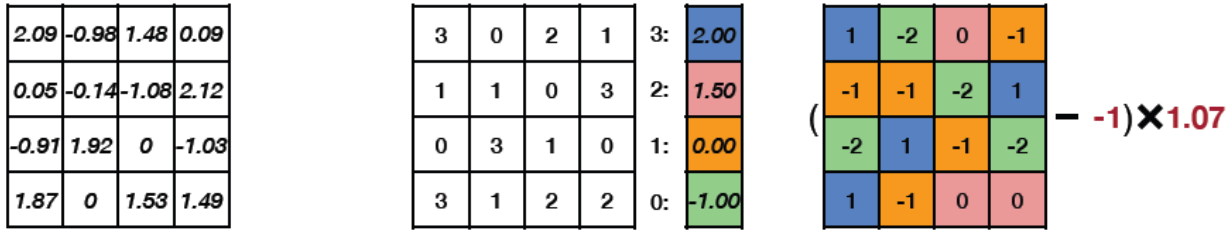
S	E	E	M
0	0	0	1
0	1	1	1

FP4 (E3M0)

S	E	E	E
0	0	0	1
0	1	1	1

Métodos de Despliegue Eficiente

- **Cuantización de pesos. Métodos:**

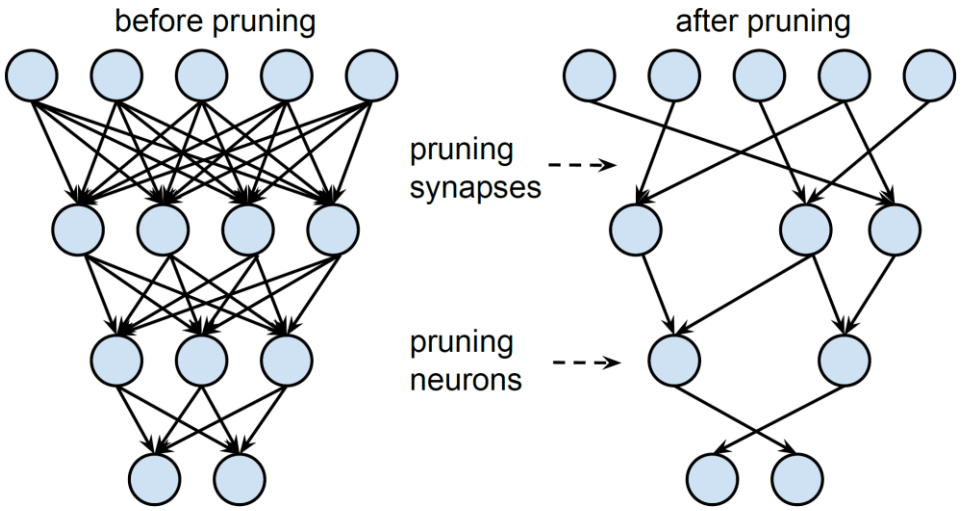


		K-Means-based Quantization	Linear Quantization	Binary/Ternary Quantization
Storage	Floating-Point Weights	Integer Weights; Floating-Point Codebook	Integer Weights	Binary/Ternary Weights
Computation	Floating-Point Arithmetic	Floating-Point Arithmetic	Integer Arithmetic	Bit Operations

[S. Han, efficientML.ai]

Métodos de Despliegue Eficiente

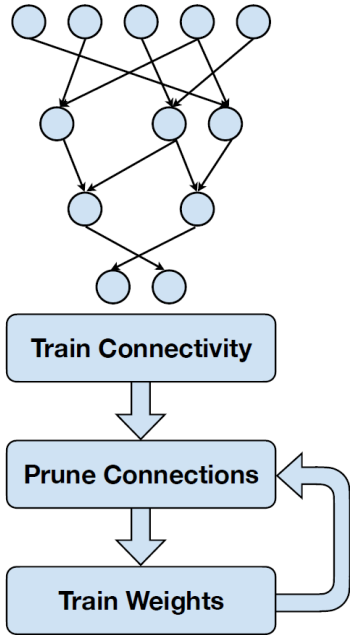
- **Poda (Pruning):**
  - Después de cuantización, muchos pesos igual a 0
  - En algunas redes, es posible incluso eliminar hasta el 90% de los pesos sin perder accuracy

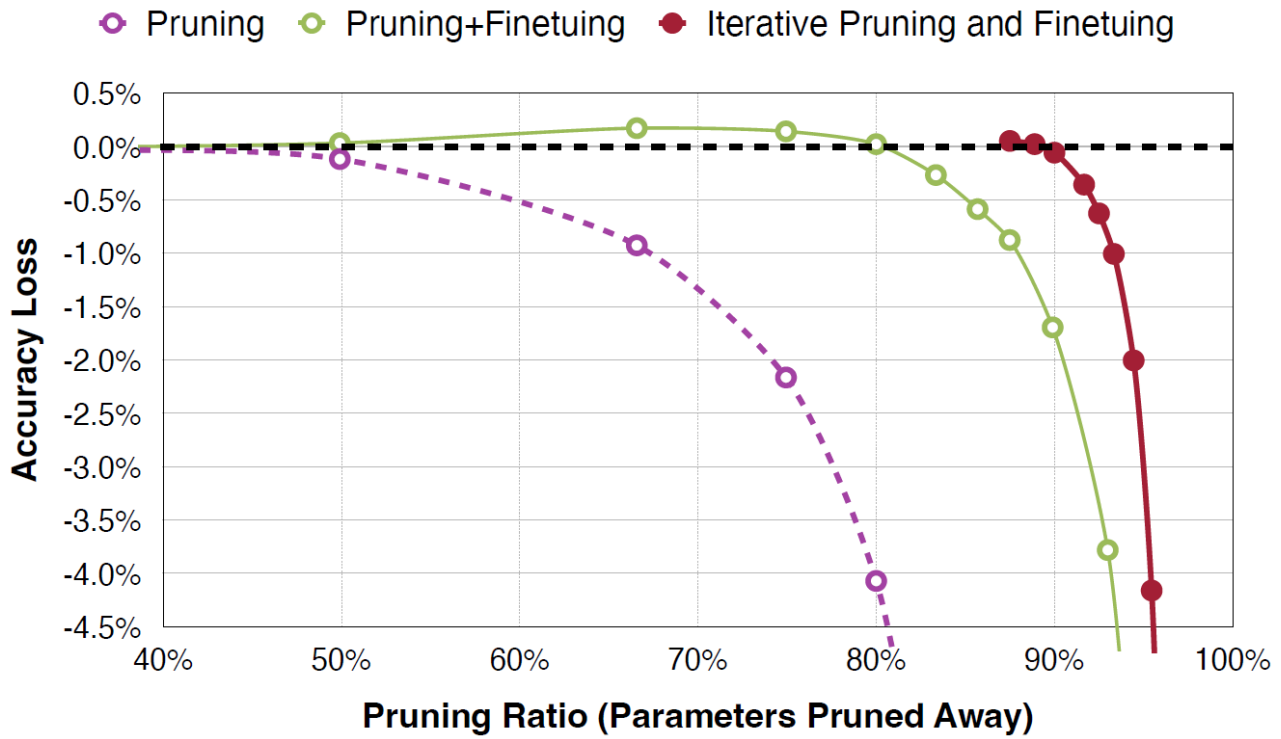


[LeCun et al, Optimal Brain Damage, NeurIPS 1989]

Métodos de Despliegue Eficiente

- **Pruning. Metodología:**

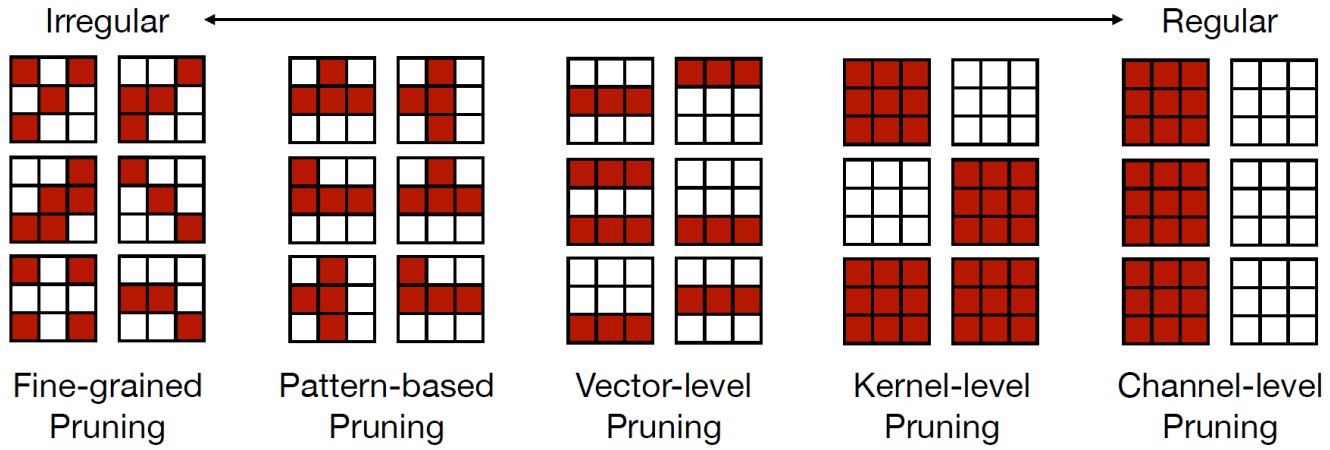


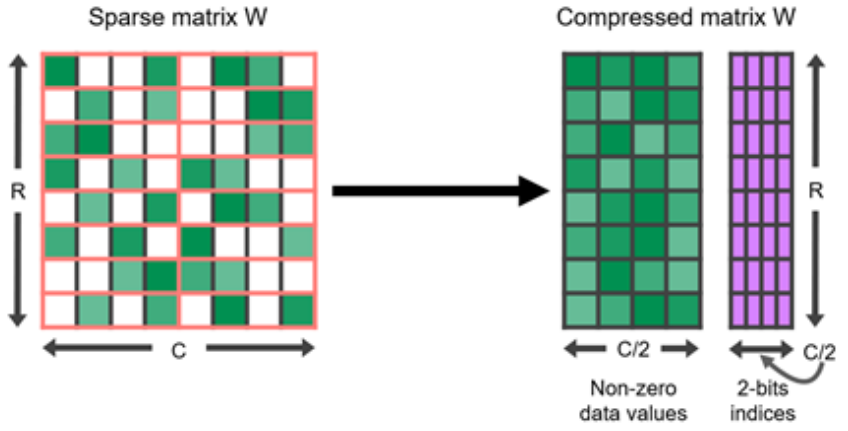


[S. Han et al, Learning both Weights and Connections for Efficient Neural Networks, NIPS 2015]

Métodos de Despliegue Eficiente

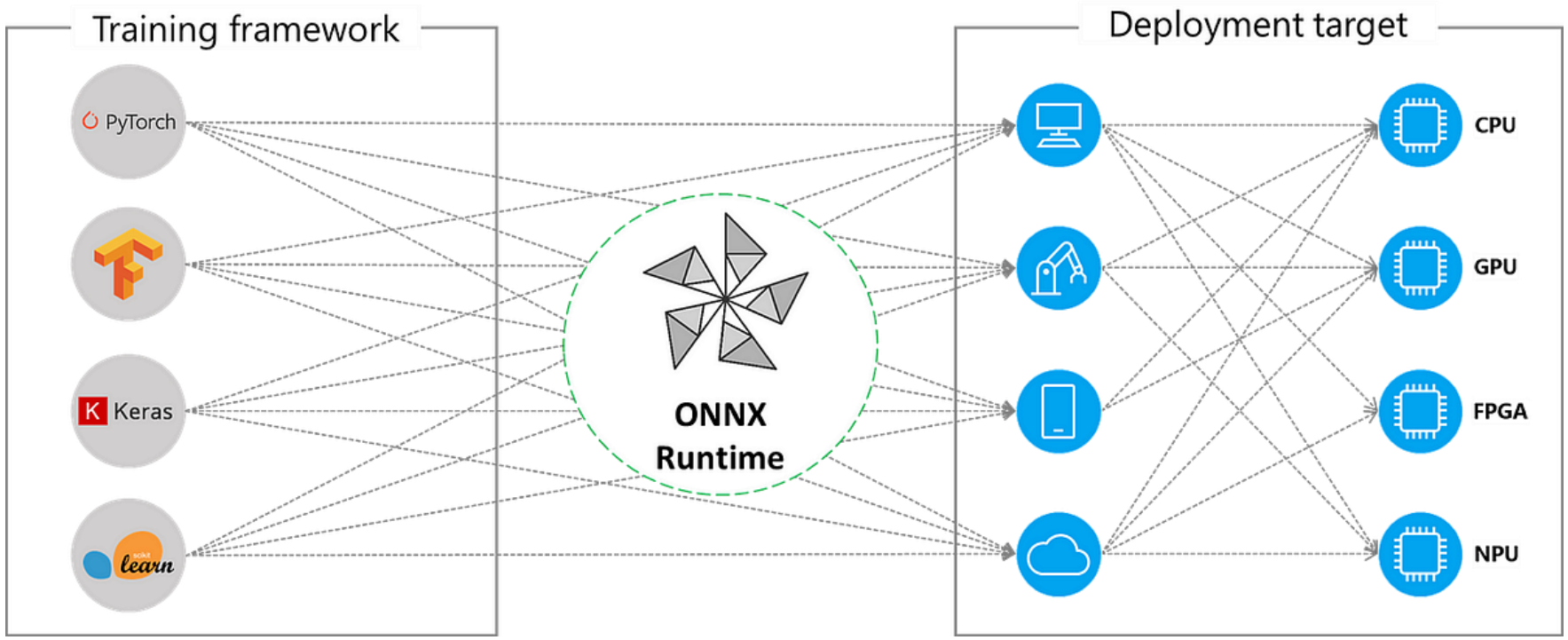
- **Patrones de dispersión (Sparsity):**
  - Podemos hacer pruning a distintos niveles:
    - Más Irregular: menos pérdida de precisión pero menos paralelismo (eficiencia)
    - Más Regular: más pérdida de precisión pero más paralelismo (eficiencia)
  - Desde las GPUs Ampere de NVIDIA, podemos manejar matrices dispersas





Métodos de Despliegue Eficiente

- **ONNX: formato estándar**



Métodos de Despliegue Eficiente

- **PyTorch**:
  - Optimizar modelos con: *ASP* (Automatic SParsity), *torch.nn.utils.prune* y *torch.quantization*
  - Despliegue eficiente: *torchao*
- **Tensorflow**:
  - Optimizar modelos con: *tensorflow_model_optimization*
  - Despliegue: *TFLite*
- **NVIDIA TensorRT:** optimizador de modelos para inferencia que soporta PyTorch, Tensorflow y ONNX
- **Servidores de inferencia:** NVIDIA Triton, TorchServe, Tensorflow Serving, ONNX Runtime

Recapitulación

- Hemos repasado de nuevo el **proceso** a seguir para entrenar un modelo desde cero.
- Hemos visto algunos métodos para **optimizar** los **hiperparámetros**:
  - Prueba y error, o **babysitting**
  - Paralelos: en **grid, random** o **búsqueda evolutiva**
- Para hacer un despliegue, suele convenir optimizar el modelo:
  - **Cuantización**
  - **Pruning**
- Uso del formato ONNX para interoperabilidad