Pytorch

Siham Tabik

Dpto. Ciencias de la Computación e I.A.

Universidad de Granada

siham@ugr.es

UNIVERSIDAD DE GRANADA

Esquema

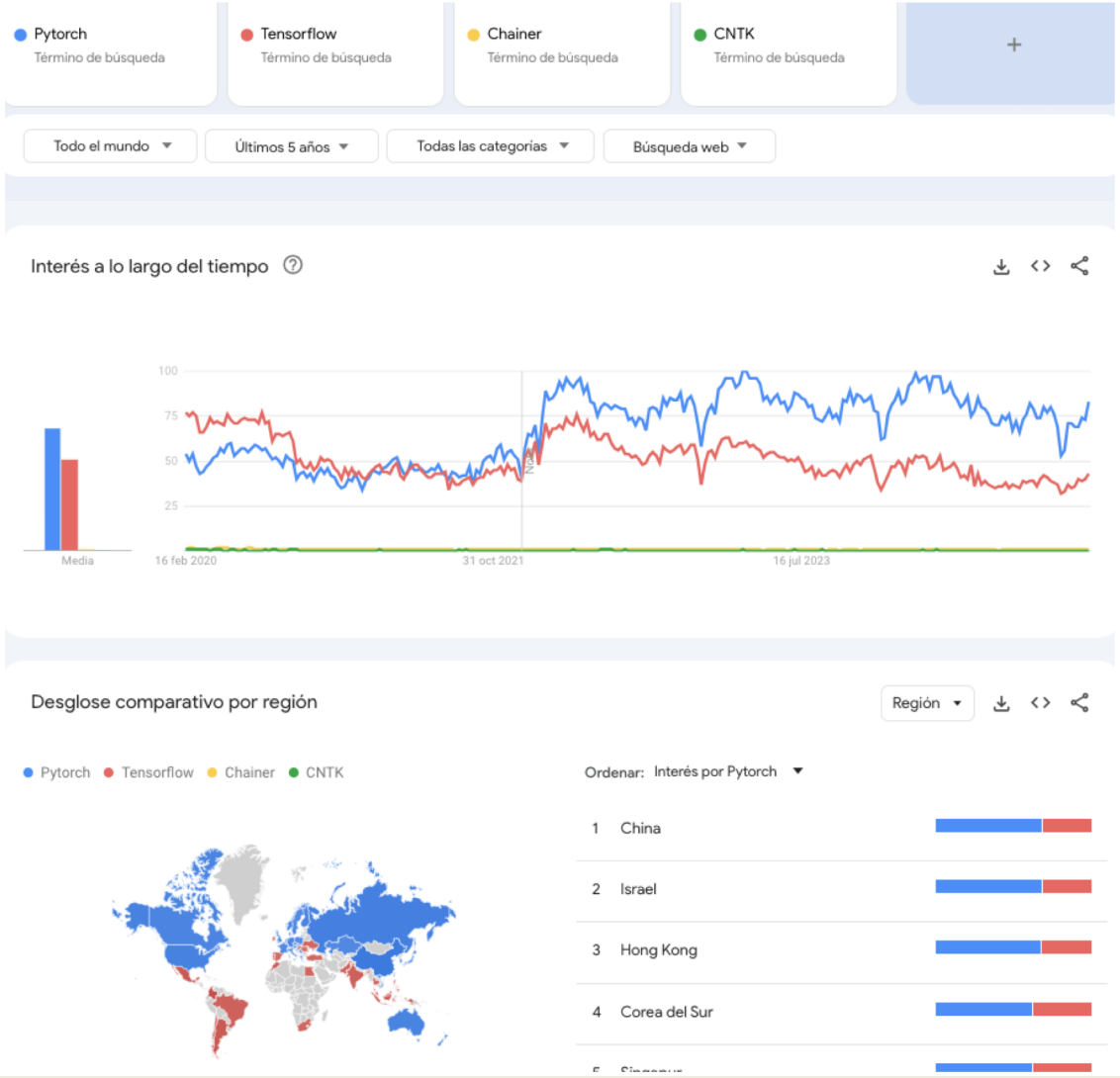
- Librerías para programar redes neuronales
- ¿Cual es son las más populares y en qué se diferencian?
- Pytorch versus Tensorflow
- Principales modulos
- ¿Cómo estructurar un Código Pytorch?
- Tipico Código Pytorch
- Ejemplos

¿Por qué no hacerlo uno mismo?

- Difícil de comprar con los modelos existentes
- Difícil de separar modelo y optimización
- Más difícil de depurar
- Difícil de extender y retocar el modelo

Librerías para programar Redes Neuronales

Popularidad según google trends



¿En qué se diferencian estas librerias?

- La principal diferencia es –independientemente del código- la forma en que crean y ejecutan los cálculos.
- En general, representan las redes neuronales como grafos computacionales (grafo acíclico dirigido o DAG): Nodos, aristas
- Existen dos tipos de grafo:
  - **Grafo estático**: se crean y se conectan todas las variables al principio y se inicializan en una sesión estática (inmutable). Esta sesión y este grafo persisten y se reutilizan: no se reconstruyen después de cada iteración de entrenamiento, lo que lo hace eficiente.
  - **Grafo dinámico**: se construye dinámicamente, inmediatamente después de declarar las variables. Este grafo se reconstruye **después de cada** iteración de entrenamiento. Los grafos dinámicos son flexibles y nos permiten modificar e inspeccionar el interior del grafo en cualquier momento. El principal inconveniente es que puede llevar tiempo reconstruir el gráfico.
  - Pueden ser más eficientes dependiendo de la aplicación específica y la implementación.

Tensorflow versus Pytorch

Feature	TensorFlow-Keras	PyTorch
Execution Model	Static Graph (default) + Eager Execution	Dynamic Graph (default) + static mode
Syntax	Higher-level, easier for beginners	Pythonic, intuitive for researchers
Deployment	Strong support (TF Serving, TF Lite)	TorchScript, TorchServe improving
Performance	Optimized for large-scale production	Great for research, improving in production
Community	More industry adoption	More research adoption

¿Qué es Pytorch?

- Es un ecosistema de herramientas, librarías y recursos para accelerar la construcción y despliege de los model de ML
- El usuario/programador se puede centrar en seleccionar y asemblar los modulos/bloques pre-fabricados (una capa conv es un bloque, una capa FC es otro bloque, loss functions, optimizers)
- El calculo de los gradientes es automático
- Considerado como el más facil de usar en el ámbito de investigación
- Existe muchos recursos, e.g., tutorials, videos, códigos, entornos (colab)
- Open-source, gran comunidad (+1,200 colaboradores, 50% Year-Over-Year growth, 22K Foros de usuarios)
- Se usa en cursos de ML en muchas universidades: Berckley, Stanford, Toronto
- El 70% de los códigos publicados en paper with code están escritos usando Pytorch

NOTE: Latest PyTorch requires Python 3.9 or later.

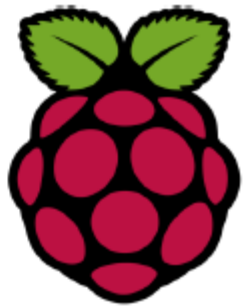
Pytorch es Flexible

Desde Raspberry Pi, Android, Windows, iOS, Linux

PyTorch Build	Stable (2.5.1)			Preview (Nightly)	
Your OS	Linux		Mac		Windows
Package	Conda	Pip		LibTorch	Source
Language	Python			C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.4	CUDA 12.6	ROCm 6.3	CPU
Run this Command:	`pip3 install --pre torch torchvision torchaudio --index-url https://download.pytorch.org/whl/nightly/cpu`				

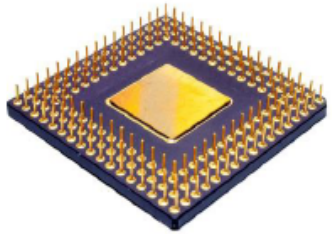




Pytorch es Protable

Desplegar los códigos en uno o más CPUs o GPUs de un sobremesa, servidor o smartphone con TorchScript , PyTorch Lightning, TorchServe



CPU

GPU

Android

iOS

Raspberry Pi

Principales módulos

torch

[Pytorch's main package, contains data structures to acomodate tensors and operate with tensors]

torchvision

[provides popular datasets, transforms and models for CV]

nn

[contains all necesary clases to build NN, conv layers, linear layers, activation functions, pooling layers, loss functions,…]

functional

[same features as torch.nn but in functional form]

nn.Module

[a container that all modelos should inherit from because it tracks the trainable parameters]

utils

[provides classes for manipulating, e.g. Dataset and Dataloader]

datasets

dataloader

optim

[provides optimization algortihms, SDD, Adam,…]

autograd

[automatic differentiation for all operations with tensors]

Módulos para acelerar el código

torch

cuda

[enables computational acceleration on NVIDIA GPU]

cpu

[implements abstractions found in torch.cuda to facilitate writing device-agnostic code.]

mps

[enables computational acceleration using Apple Metal Performance Shaders (MPS) on Apple GPUs]

xpu

[enables computationl acceleration using XPU on heterogenous systems, Intel GPU, CPU, FPGA]

distributed

[enables distributed computation on distributed systems, e.g., multi-GPU systems]

Cómo acelerar nuestro código?

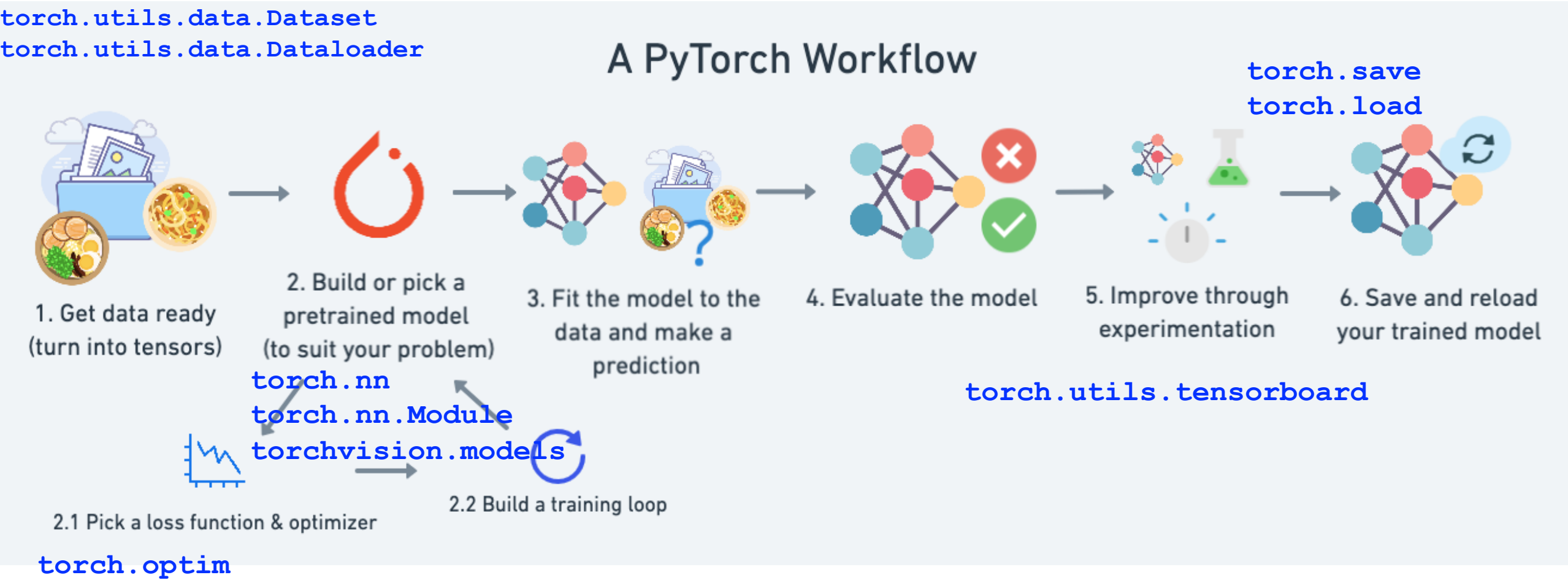
Mover el modelo y datos al device
Por ejemplo:

```
# Get cpu, gpu or mps device for training.
device = (
"cuda"
if torch.cuda.is_available()
else "mps"
      if torch.backends.mps.is_available()
      else "cpu"
)

model.to(device)
inputs, labels = inputs.to(device), labels.to(device)
```

Pytorch code workflow

`torch.utils.data.Dataset`
`torch.utils.data.Dataloader`



`torch.save`
`torch.load`

`torch.nn`
`torch.nn.Module`
`torchvision.models`

`torch.utils.tensorboard`

`torch.optim`

Típico código de pytorch

1. Cargar los datos
2. Definir el modelo
3. Especificar los parámetros del modelo
4. Especificar el bucle para entrenar el modelo
5. Evaluar el modelo

```
import torch.nn as nn

# Define a simple neural network
class SimpleNN(nn.Module):
            def __init__(self):
            super(SimpleNN, self).__init__()
             # Fully connected layer: 2 inputs, 4 outputs
            self.fc1 = nn.Linear(2, 4)
             # Fully connected layer: 4 inputs, 1 output
            self.fc2 = nn.Linear(4, 1)

            def forward(self, x):
                        # Apply ReLU activation
                        x = torch.relu(self.fc1(x))
                        # Output layer
                        x = self.fc2(x)
                        return x

# Instantiate the model
model = SimpleNN()
print(model)
```