Introducción a la librería Pytorch

Diego Andrade Canosa

Roberto López Castro

Índice

- Introducción al curso
- Conceptos básicos de Redes de Neuronas Profundas (repaso)
- Introducción a Pytorch
- Lab1: Entorno del curso
- Referencias y materiales adicionales

Introducción al curso

Introducción al curso

- Curso de Introducción a Pytorch (no a Machine Learning)
- Impartido por:
  - Diego Andrade Canosa (CITIC+UDC)
  - Roberto López Castro (CITIC+UDC)
  - Miembros de un grupo de arquitectura de computadores y HPC
  - Usamos Pytorch para investigación sobre la confluencia del HPC con ML
    - AutoML+HPC
    - Performance-aware pruning techniques and kernels

Introducción al curso

- Duración (12 horas)
  - 2, 3 y 4 de julio de 9 a 13
- Contenidos básicos:
  - Día 1:
    - Conceptos clave de Pytorch: Tensores, Variables, Gradientes, Autograd
    - Conjuntos de datos y cargadores de datos
  - Día 2:
    - Creación y composición de la arquitectura de red
  - Día 3:
    - Carga y almacenamiento de modelos
    - Casos de uso de Pytorch

Introducción al curso

- Conceptos clave de Pytorch (Diego) **DÍA 1**
  - Tensores
    - Transformaciones
  - Variables y gradientes
  - Diferenciación automática
- Conjuntos de datos y cargadores de datos (Roberto)
- Creación y composición de la arquitectura de red (Diego) **DÍA 2**
  - Los módulos nn.Module y nn.Sequential de Pytorch
  - Tipos de capas en Pytorch
  - Optimizadores y funciones de pérdida
  - Bucles de entrenamiento
    - Pasada Forward
    - Backpropagation
- Carga y almacenamiento de modelos (Roberto) **DÍA 3**
- Casos de uso (Diego y Roberto)
  - Redes CNN
  - Redes RNN
  - Transformers

Conceptos básicos de RNP

Conceptos básicos RNP

- *Artificial Intelligence* (AI) es una disciplina que basada en emular, mediante un computador, la inteligencia humana
- *Machine Learning* (ML) es una rama de la Inteligencia Artificial (IA) que permite aprender a realizar una tarea que requiere cierto grado de inteligencia sin una programación explícita de la misma
  - Para ello se utiliza el concepto de Red de Neuronas (RN) artificiales
    - Concepto fundacional: perceptrón multicapa
  - Por ejemplo, no se consideran ML los sistemas expertos basados en reglas
- *Deep Learning* (DL) es un subconjunto de ML que usa redes profundas (con muchas capas ocultas)
  - Permite procesar datos no estructurados
  - Extrae patrones de los datos
  - Basado en *representation learning*

Razones del auge del DL

- El Perceptrón Multicapa, por ejemplo, data de 1969
- El auge actual data de:
  - Año 1990: Se acuña el concepto de DL -> https://people.idsia.ch/~juergen/deep-learning-miraculous-year-1990-1991.html
  - Año 2009: Artículo de Imagenet -> https://ieeexplore.ieee.org/document/5206848
- Razones del auge:
  - Big data
  - HPC Hardware: GPUs, Supercomputers, Clusters, Cloud
  - Software accesible: Google Colab, Kaggle, Tensorflow, Pytorch

El proceso de *ML*

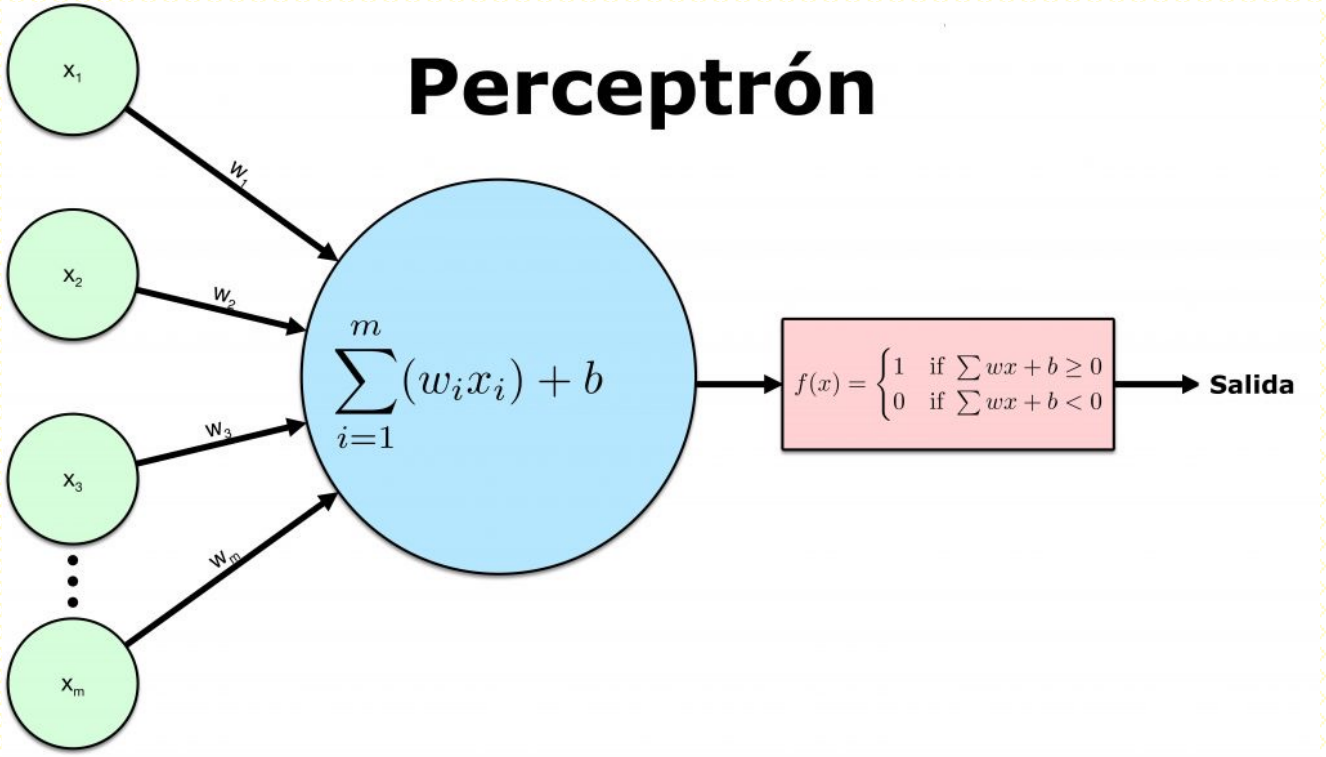
- Debemos entender el problema que queremos resolver
  - Qué salida proporcionará el modelo
  - En base a qué entradas
- Lo anterior nos permitirá definir…
  - Los datos que consideramos relevantes para el modelo
  - La representación más apropiada de estos datos

El proceso de *representation learning*

- Con *representation learning* se pueden aprender:
  - Las características (*features*) relevantes de las entradas
  - Cómo relacionar esas características con cada una de las salidas posibles

El perceptrón

Forward propagation



Fuente: http://blog.josemarianoalvarez.com/2018/06/10/el-perceptron-como-neurona-artificial/

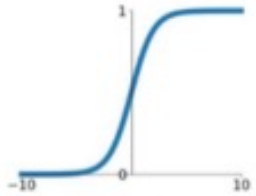
citic

Funciones de activación

Activation Functions

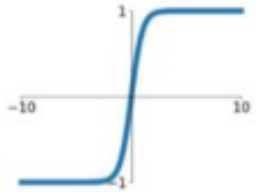
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



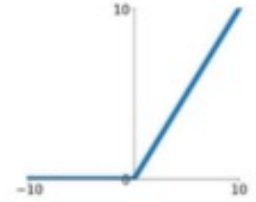
tanh

$$\tanh(x)$$



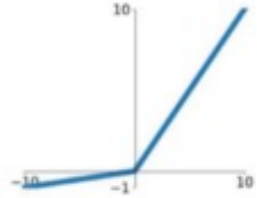
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

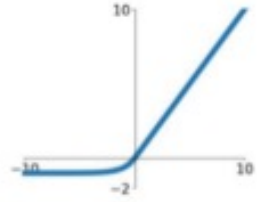


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Source: https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092

citic

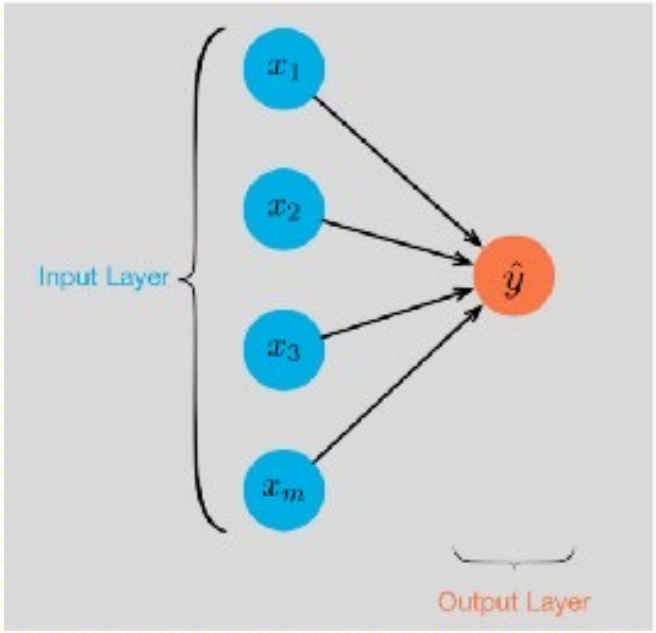
Funciones de activación

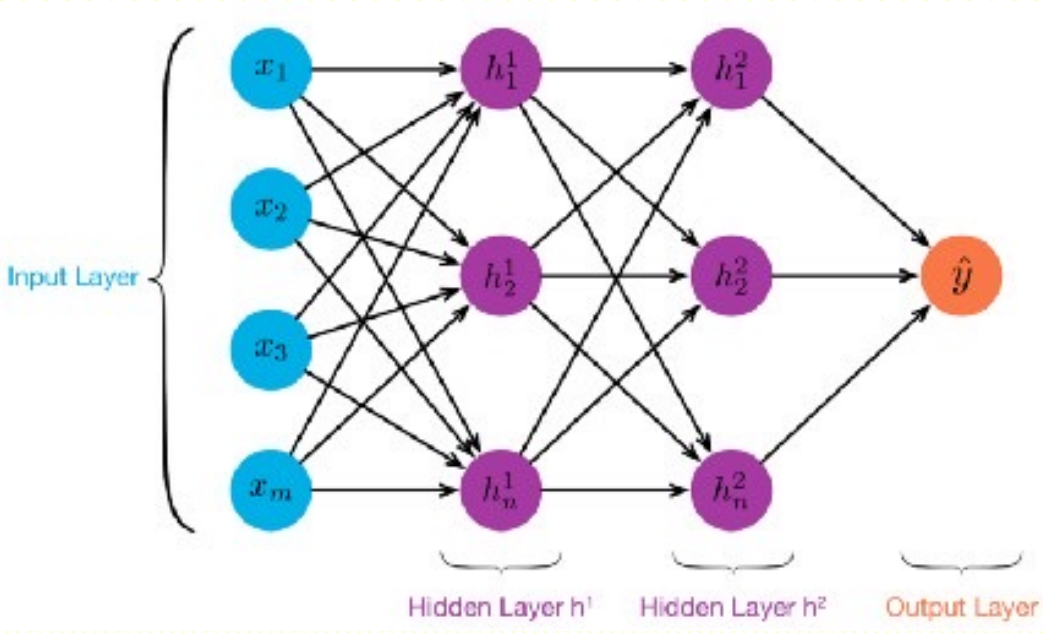
- Sigmoid: Rango de salida [0,1]. Problema *vanishing gradient*
  - Tanh: Rango de salida [0,1] Valores más próximos a 0. *Vanishing gradient*.
- ReLU *(Rectified Linear Unit):* Rango de salida [0,n]. Es lineal y no satura. Atenúa *vanishing gradient*, pero tiene el problema de *dying RELU*
- Leaking RELU: Rango de salida [-1,n]. Solventa el problema del *dying RELU*, proporcionando una salida pequeña negativa cuando el valor es negativo
- SoftMax: Rango de salida [0,1]. Se utiliza en la salida de problemas de clasificación. Convierte las salidas en probabilidades que suman 0

Funciones de activación

- Problemas comunes:
  - *Vanishing gradient:* En redes profundas, al realizar la *backpropagation* los gradientes pueden ser muy pequeños en las primeras capas
    - Problema común de activaciones que manejan rangos de valores pequeños ej. [0,1]
  - *Dying RELU:* Cuando se usan activaciones con rangos de salida amplios [0,n], se podrían generar gradientes altos que al fluir hacia atrás en la red podría "matar" neuronas
    - Las neuronas podrían no volver a activarse nunca más.
      - Tienen valores muy negativos y la salida de la activación es 0, lo cual provoca gradientes nulos
      - Por eso, activaciones con salidas [-1,n] atenúan el problema

Perceptrón multicapa





Fuente: https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

citic

Componentes básicos de un modelo de DL

- Tres bloques de capas identificables:
  - Capa de entrada
  - Capas ocultas (intermedias)
  - Capa de salida
- Cada capa está compuesta de neuronas, interconectadas entre si, cuya señal se caracteriza por:
  - Peso
  - *Bias*
  - Función de activación

Conceptos clave: Repaso

- Inferencia, Entrenamiento
- Conjunto de datos (d*ataset)*:
  - Conjunto de entrenamiento
  - Conjunto de validación
- Función de activación
- Función de pérdida
  - Optimización de la función de pérdida
    - Algoritmos de *Gradient Descent*
      - SGD, Adam, Adadelta, Adagrad, RMSProp

Otros conceptos a revisar relacionados con el entrenamiento

- *Minibatches*
  - Asociado al concepto de *Stochastic Gradiente Descent* (SGD)
  - Los gradientes de los pesos se calculan procesando a la vez *minibatches* de elemento del conjunto de datos
  - Permite acelerar el entrenamiento
    - Mediante paralelización
- *Learning rate*
  - Son adaptativos a la pendiente observada
- *Overfitting*
- Regularización
  - Mejora la capacidad de generalización de nuestro modelo
  - Mejora el rendimiento de nuestro modelo en datos no vistos
    - Su capacidad de generalización

Batches y minibatches

- *batch_size=total_samples* -> El conjunto de datos se procesa por completa antes de actualizar los pesos
  - Batch Gradient Descent
- *batch_size*=n -> El conjunto de entrenamiento se procesa en batches de n *samples*. Después de procesar cada batch se actualizan los gradientes
  - Minibatch Gradient Descent
- *batch_size*=1 -> Los gradientes se actualizan después de procesar cada *sample*
  - *Stochastic Gradient Descent*

Epoch y step

- Cada procesado del conjunto de datos completo se conoce como *epoch*
- El procesado de los *batch_size* samples requeridos para actualizar los gradientes, produce que una *epoch* se divida en *total_samples/batch_size*

Métodos de regularización

- *Dropout*: Técnica que consiste en poner algunos pesos a cero aleatoriamente
  - Alrededor del 50%
- *Early Stopping:* Parar un entrenamiento prematuramente
  - Una forma de evitar overfitting

Por tipo de aprendizaje

- Aprendizaje supervisado: El sistema se entrena con un conjunto de datos que contiene pares formados por una entrada y su correspondiente salida correcta (*ground-truth*)
- Semi-supervisado: Una parte del conjunto de datos de entrenamiento está etiquetado y otra no
  - La parte no etiquetada se etiqueta usando el modelo ya entrenado con la parte etiquetada
- Aprendizaje no-supervisado: El conjunto de entrenamiento está compuesto de datos no etiquetados
  - Concepto de clustering
- Aprendizaje auto-supervisado: El conjunto de entrenamiento usa datos no etiquetados. Debe usar técnicas creativas para identificar la ground-truth. Ejemplos:
  - Denoising: Se usan imágenes completas, se corrompe una parte de la imagen, y se usa el conocimiento de la imagen original (sin corromper) para entrenar el modelo
  - Autoregresión: Se quiere predecir el siguiente valor de una secuencia, se pueden usar distintas "ventanas" de esa secuencia para predecir el siguiente valor
  - Aprendizaje de embedding: En modelos de lenguaje la representación numérica de los tokens se suele aprender y promueve que los tokens con significados similares tengan valores numéricos parecidos

Por el tipo de tarea que aprende el modelo

- Modelos de regresión: Aprenden a predecir un valor
- Modelos de clasificación: Aprenden a clasificar la entrada en una de entre varias categorías
  - Binaria: Si las categorías son 2
  - Multiclase: Si las categorías son más que 2
- Modelos de clustering: Aprenden a agrupar los datos de entrada en varias categorías, o clústeres
- Aprendizaje por refuerzo: Aprenden a tomar decisiones que se evalúan en base a una función que calcula la recompensa de la decisión adoptada

Por tipo de arquitectura de red

- Perceptrón Multicapa
- *Convolutional Neural Networks* (CNN)
- *Recurrent Neural Networks* (RNN)
  - *Long Short-Term Memory* (LSTM)
- *Generative Adversarial Networks* (GAN)
- *Autoencoders*
- *Transformers*

Clasificaciones de modelos de DL

- Por tipo de aplicación:
  - Modelado de secuencia. Predicciones (RNNs)
    - Procesamiento de lenguaje natural (LSTMs)
      - Traductores
      - Asistentes/conversadores virtuales
      - Reconocimiento de voz
  - Visión por computador (CNNs)
    - Seguimiento de objetos
    - Identificación de imágenes
    - Conducción autónoma (LIDAR)
  - Modelos generativos (Autoencoders)
    - Imágenes. *Prompt models*
    - Vídeos
    - Texto
  - …

Introducción a la librería Pytorch

Pytorch: Génesis

- Librería creada en el año 2016 por FAIR (Facebook AI Research Lab)
  - Proyecto Adam Paszke (Internship)
  - Sucesor de la librería *torch (*basada en *Lua)*
  - Influenciado por la librería *chainer*
- Cronograma
  - Año 2016: Versión inicial
  - Año 2018: Versión 1.0 (fusión con Caffe)
  - Año 2019: Pytorch Lightning
  - Año 2023 (marzo!): Pytorch 2.0

Fuente

Fortalezas

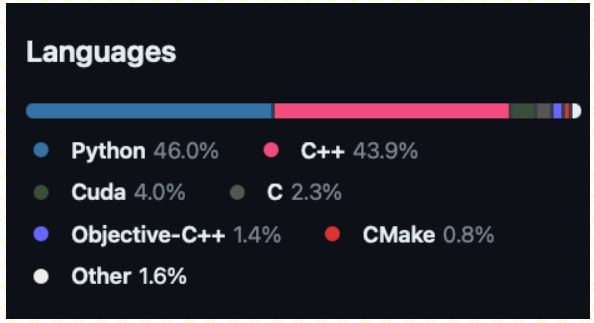
- Gran penetración en la comunidad investigadora
- Ventaja conceptual: Dynamic Computation Graph
- Facilidad de uso
- Flexibilidad
- Mucha base de código abierto en:
  - Visión por Computador
  - Procesado de Lenguaje Natural
  - Sistemas recomendadores
  - Procesado de señal

Ecosistema Pytorch

- Conjunto de herramientas y librerías que enriquecen y complementan al núcleo principal de Pytorch
- https://pytorch.org/ecosystem/
  - Domain-specific
    - NeMO
    - Diffusers
    - PennyLane
    - Transformers
  - HPC
    - Lightning
    - DeepSpeed
    - FairScale https://github.com/facebookresearch/fairscale
    - Accelerate
    - Ray
  - Other
    - TorchMetrics

Principios de diseño

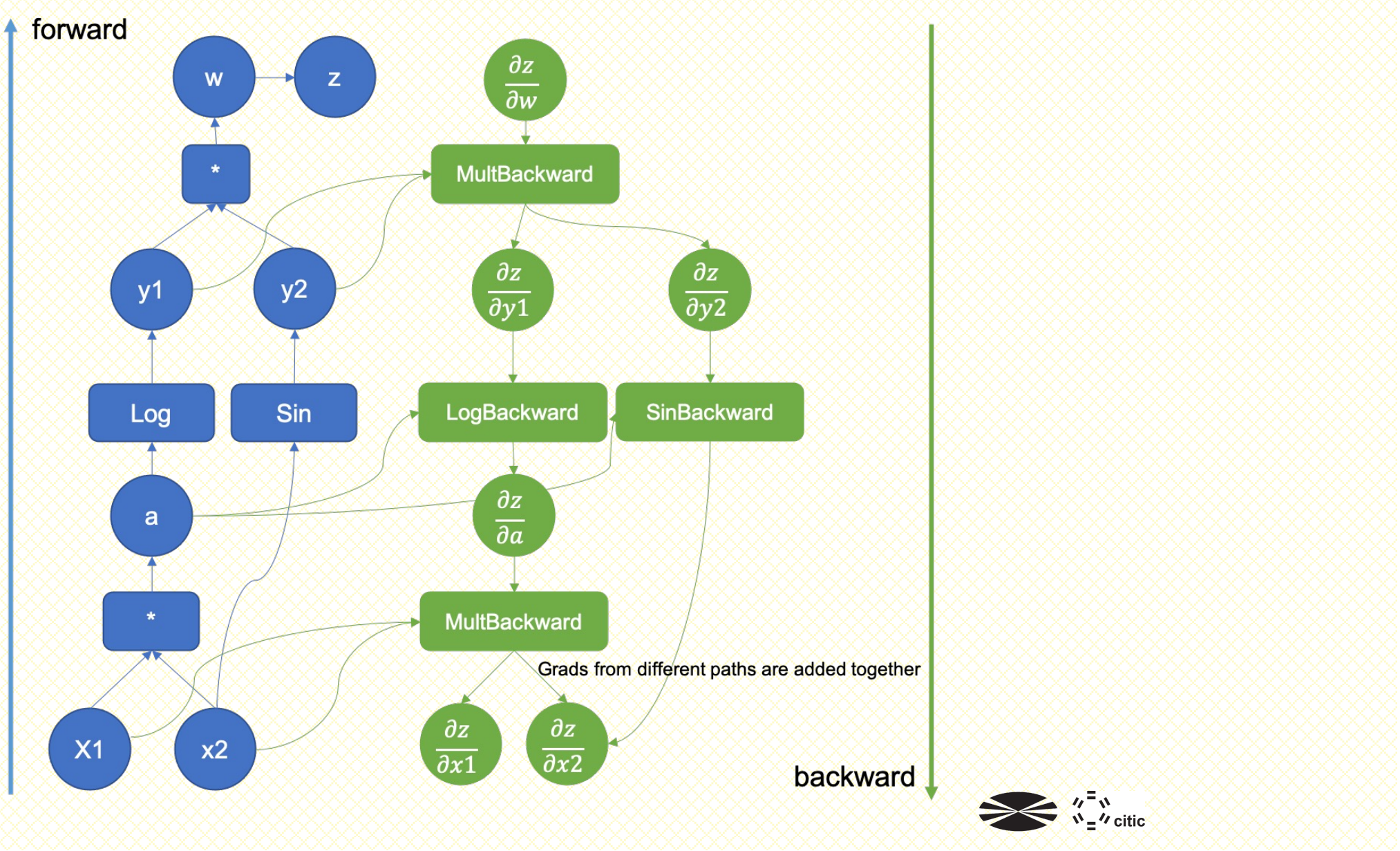
- Principio 1: Usable antes que rápido
- Principio 2: Simple antes que fácil
- Principio 3: Python primero



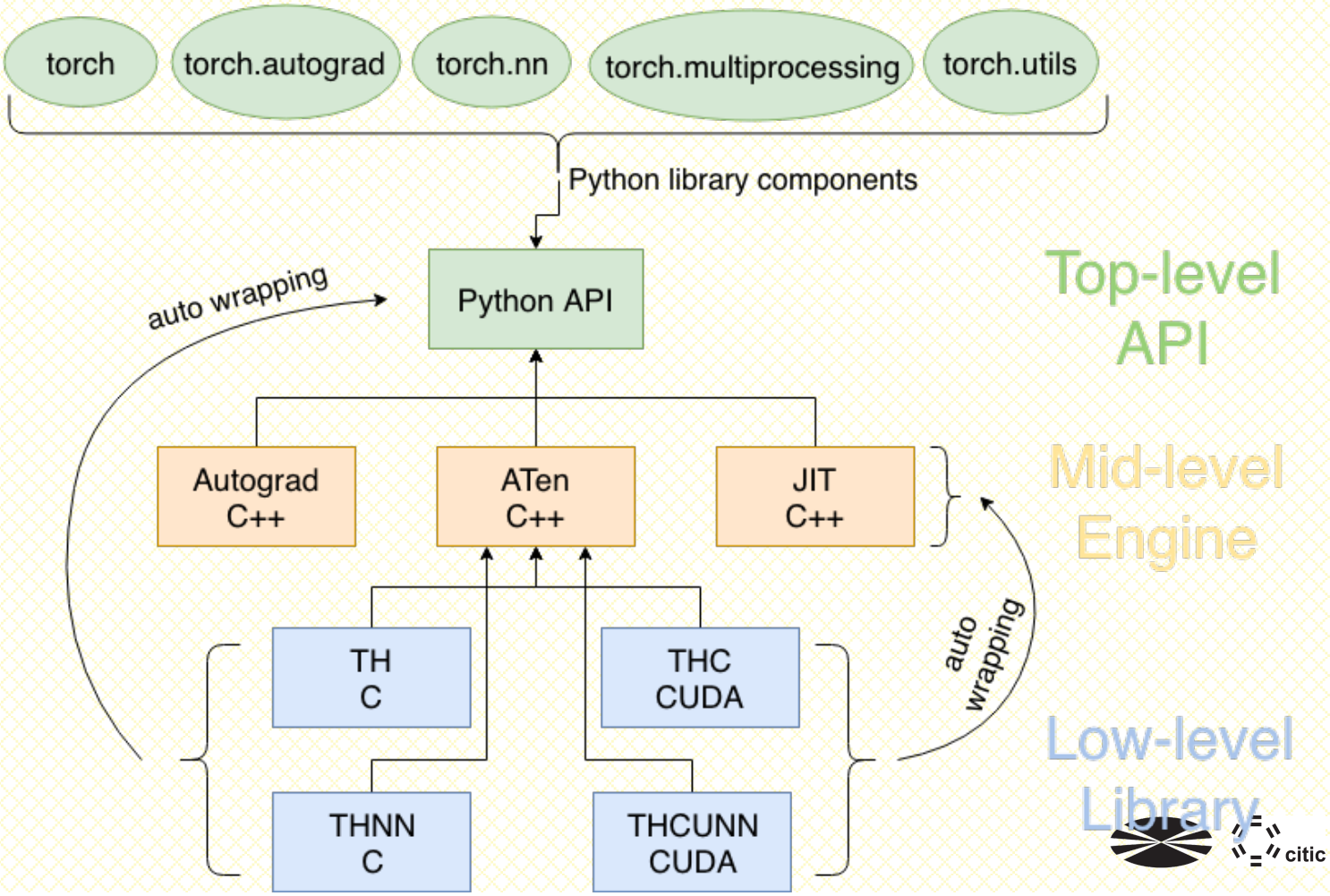
citic

Dynamic Computation Graph (DCG)

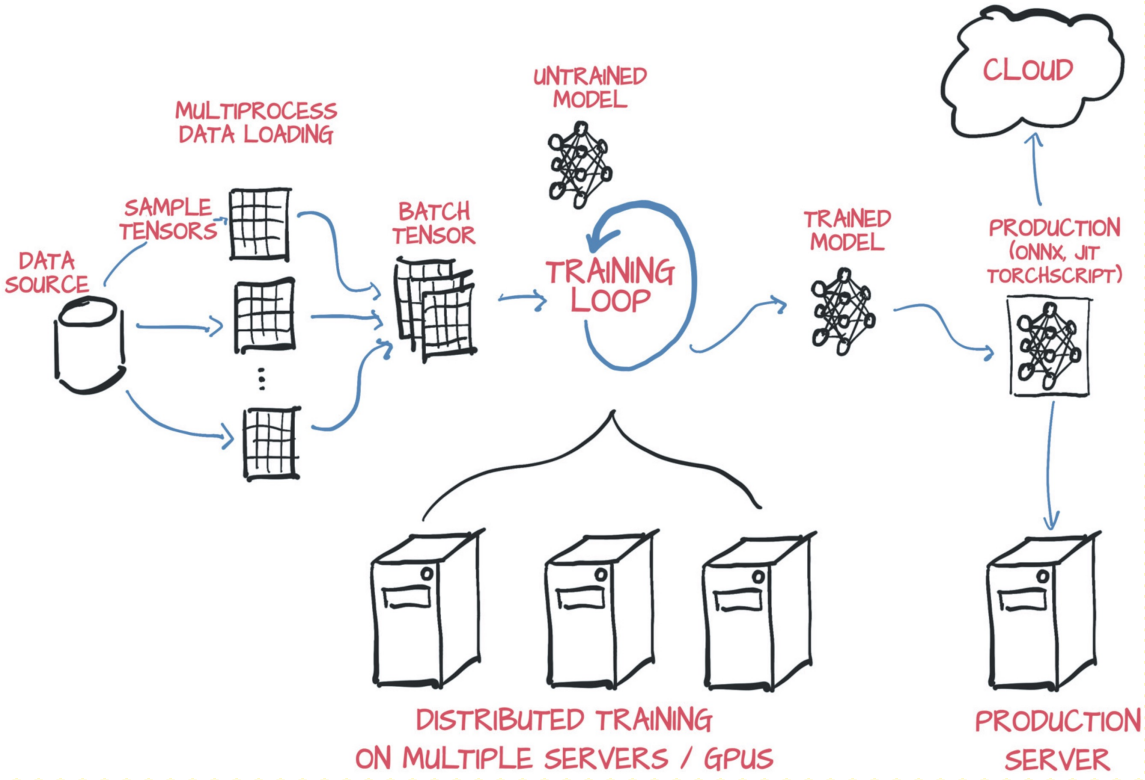
- Aproximación dinámica “define by run”, (vs “define and run”, estática)
- Más intuitivo y flexible que la aproximación estática (static computation graph)
- Menos posibilidades de optimización del cómputo
- Es una diferencia básica respecto a TF (<2.0)


forward
w
z
*
y1
y2
Log
Sin
a
*
X1
x2
$\frac{\partial z}{\partial w}$
MultBackward
$\frac{\partial z}{\partial y1}$
$\frac{\partial z}{\partial y2}$
LogBackward
SinBackward
$\frac{\partial z}{\partial a}$
MultBackward
Grads from different paths are added together
$\frac{\partial z}{\partial x1}$
$\frac{\partial z}{\partial x2}$
backward
citic

Estructura de módulos de Pytorch



Pytorch: Uso



citic

Abstracciones

- Tensores: estructura de datos multidimensional donde se almacena los diversos elementos del modelo: parámetros (pesos), bias, entradas, salidas
- Contenedores: estructuras a las que se asocian los distintos componentes de un modelo, así como su operación
- Optimizadores: artefacto que permite calcular los gradientes de los pesos en función de la salida de la función de pérdida
- Autograd: mecanismo que crea el DCG y permite calcular las derivadas parciales

Bloques constructores

- API Python
  - torch
  - torch.nn
- Librerías secundarias
  - Torchtext (Texto)
  - Torchaudio (Audio)
  - Torchvision (Visión por computador)
  - Torcharrow (Data frame, tabular data)
  - TorchData (Pipelines de procesado de datos)
  - TorchRec (Sistemas recomendadores)
  - TorchServe (Servidores)
  - Torchx (Lanzador para aplicaciones Pytorch)

torch

- Tensores
  - Definición, Creación, Indexación, Transformación, Random sampling, Serialización, Operadores matemáticos (aritméticos, lógicos, espectrales, Blas/lapack, operaciones foreach, otros)
- Utilidades
- Optimizaciones
- Configuración del engine

https://pytorch.org/docs/stable/torch.html

torch.nn

- Definición de capas
  - Convolucionales, Pooling, Padding, Normalization, Recurrent, Dropout...
- Contenedores (containers)
  - Module, Sequential
- Funciones
  - Distance, Loss, Quantized

https://pytorch.org/docs/stable/nn.html

TorchVision

- Paquete orientado a la resolución de problemas de visión por computador
- Proporciona:
  - Técnicas para transformar y aumentar imágenes (Geometría, color, composición, conversión, auto-augmentation)
  - Datapoints (Imágenes, Vídeo)
  - Modelos (RESNET, Mobilenet, Yolo, Inception, VGG, etc...)
  - Datasets (CIFAR, MNIST, ...)
  - Utilidades
  - Entrada/Salida (para imágenes y vídeos)
  - Extracción de características

TorchAudio

TorchAudio

- Paquete que agrupa varias utilidades relacionadas con el procesamiento de audio y de señal en general
- Proporciona:
  - Modelos (Conformer, HuBERT, DeepSpeech, etc...)
  - Datasets (CMU, GTZAN, LibriSpeech, etc...)
  - Pipelines: permiten usar modelos preentrenados para realizar tareas específicas
  - Componentes para la Entrada/Salida (StreamReader, StreamWriter, play_audio)

TorchText

TorchText

- Paquete que agrupa varias utilidades realizadas con el procesamiento de lenguaje
  - Es un desarrollo incipiente basado en la filosofía de otros paquetes más longevos como torchaudio y torchvision
- Proporciona:
  - Modelos (Roberta)
  - Utilidades (extraer archivo, descargar de url,...)
  - Transformaciones
  - Vocab (mapeado de palabras a índices)
  - Datasets
  - Elementos funcionales
  - Contenedores específicos (MultiheadAttentionContainer)

TorchArrow

- Conjunto de utilidades para el procesado de datos tabulares (Data Frame)
- Proporciona:
  - DataFrame (Creación, Inspección, Transformación, Estadísticas, Artimética)
  - Column: Estructura de datos unidimensional con datos de un único tipo de datos (numérico, string, list)

TorchData

- Paquete con componentes que permiten la construcción de pipelines para la carga de datos
- Proporciona:
  - Pipelines iterables (accesibles elemento a elemento)
  - Pipeline Map-style (accesibles clave-valor)
  - Utilidades
  - DataLoader2: Versión alternativa de torch.utils.data.DataLoader

TorchRec

- Paquete que contiene varias utilidades para construir sistemas recomendadores

TorchServe

- Paquete que contiene utilidades para la creación de servidores de modelos
- Proporciona:
  - API de gestión de modelos
  - API de inferencia
  - Soporte para distintas soluciones tecnológicas (Sagemaker, Mlflow, Kubeflow, etc...)

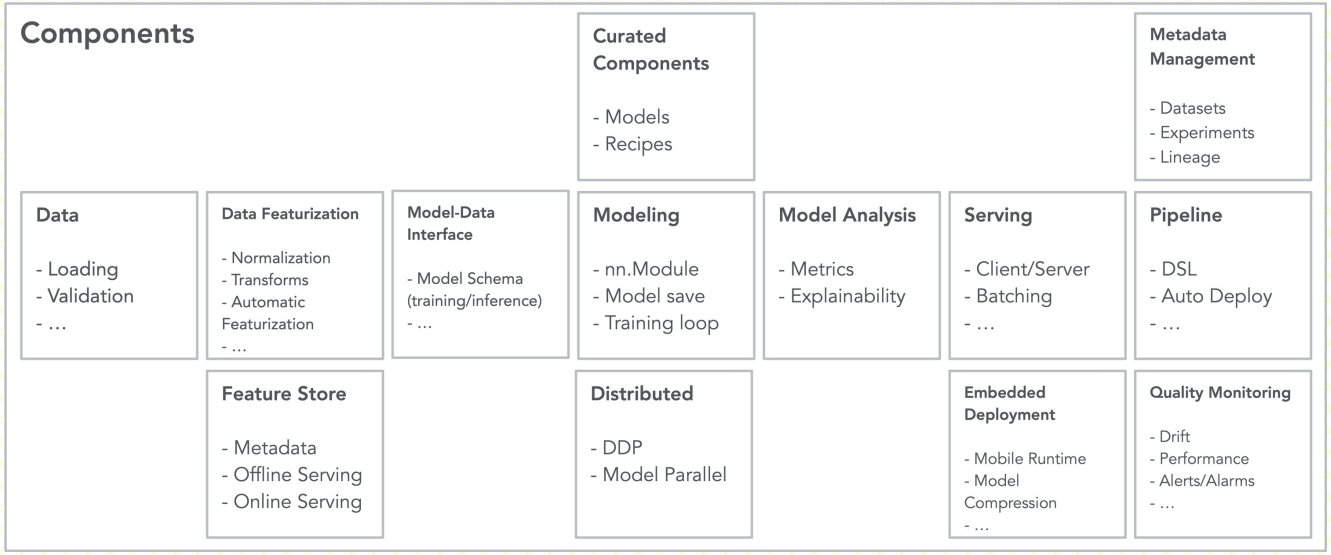
torchx

Lanzador para Pytorch

Tipo srun o qsub

Funciona con Slurm, Ray, AWS, Docker, Kubernetes, etc...

Permite ejecución local, remota y distribuida



citic

Laboratorio

- Clona el repositorio de github con los materiales prácticos del curso

  En FT3: cd $STORE

  git clone https://github.com/diegoandradecanosa/Cesga2023Courses

Laboratorio

- Instalación del entorno. Opción 1: En una instalación local de python con pip
  - Instala a través de pip los siguientes paquetes:
    - pip3 install torch==2.0.1
    - pip3 install torchvision torchtext jupyter
- Comprueba la instalación de pytorch

  python -c "import torch; print(torch.__version__)”

  python –c “import torch; print(torch.cuda.is_available()”

Laboratorio

- Instalación del entorno. Opción 2: En ft3 con conda unpack
  - Conectarse a ft3
  - Conectarse a login1 (ssh login1)
  - Desde cd $STORE

```
mkdir –p mytorch
tar –xzf /tmp/mytorch.tar.gz –C mytorch
        (Lleva tiempo)

source mytorch/bin/activate
python
conda-unpack
```

Laboratorio

- Instalación del entorno. Opción 2: En ft3 con conda
  - pip install ipykernel
  - python -m ipykernel install --user --name=mytorch
  - cd Cesga2023Courses/pytorch
  - jupyter notebook –ip `hostname –i`
  - Entrar en jupyter con el navegador local y seleccionar el kernel mytorch
  - Ejecutar la única celda del notebook bootstrap.ipynb para comprobar la instalación

Laboratorio

- Instalación del entorno. Opción 3: En ft3 con conda
  - Desde cd $STORE

    curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

    sh Miniconda3-latest-Linux-x86_64.sh

    conda config --set auto_activate_base false

    conda create –n mytorch –-solver classic

    conda activate mytorch

    conda install conda-libmamba-solver --solver classic

    compute —gpu

    conda deactivate

    conda activate mytorch

    conda install pytorch torchvision torchaudio torchtext jupyter pytorch-cuda=11.7 -c pytorch -c nvidia –-solver libmamba

Laboratorio

- Instalación del entorno. Opción 3: En ft3 con conda
  - pip install ipykernel
  - python -m ipykernel install --user --name=mytorch
  - cd Cesga2023Courses/pytorch
  - jupyter notebook –ip `hostname –i`
  - Entrar en jupyter con el navegador local y seleccionar el kernel mytorch
  - Ejecutar la única celda del notebook bootstrap.ipynb para comprobar la instalación

Referencias y materiales adicionales

Introducción al Deep Learning

- https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/
- https://mlu-explain.github.io
- https://aws.amazon.com/es/machine-learning/mlu/

Referencias

- Pytorch Internals: http://blog.ezyang.com/2019/05/pytorch-internals/
- Pytorch Design Philosophy: https://pytorch.org/docs/stable/community/design.html
- Pytorch: https://se.ewi.tudelft.nl/desosa2019/chapters/pytorch/
- A Tour of Pytorch Internals (Part I): https://pytorch.org/blog/a-tour-of-pytorch-internals-1/
- Pytorch CheatSheet: https://pytorch.org/tutorials/beginner/ptcheat.html
- Pytorch Deep Learning Framework: Speed+Usability: https://syncedreview.com/2019/12/16/pytorch-deep-learning-framework-speed-usability/