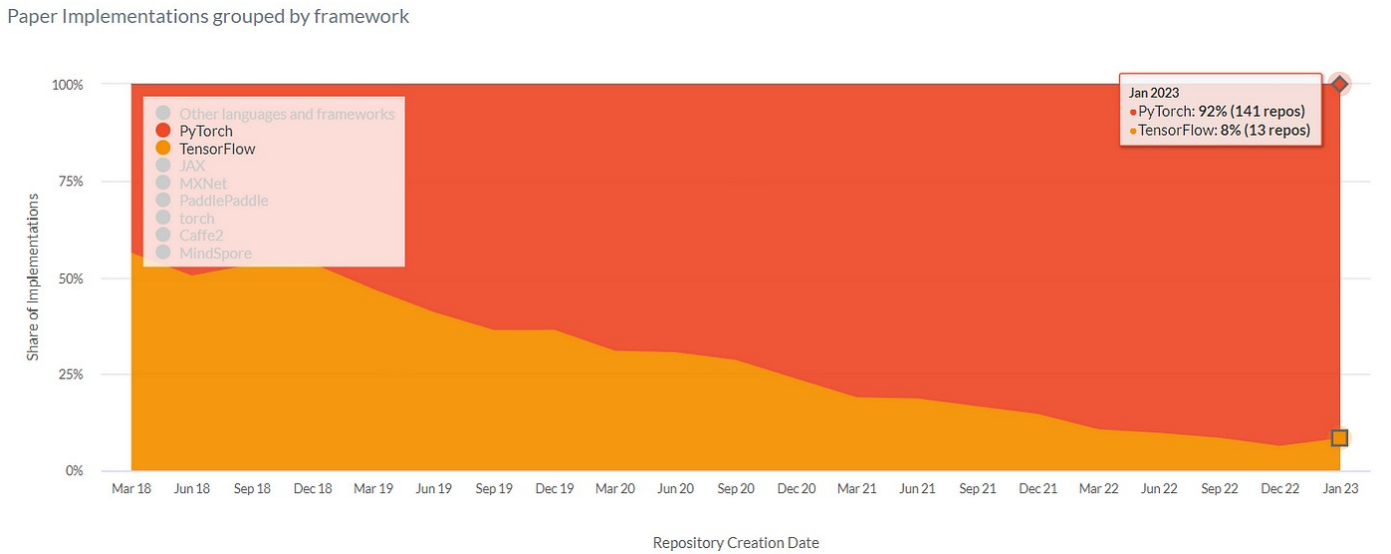
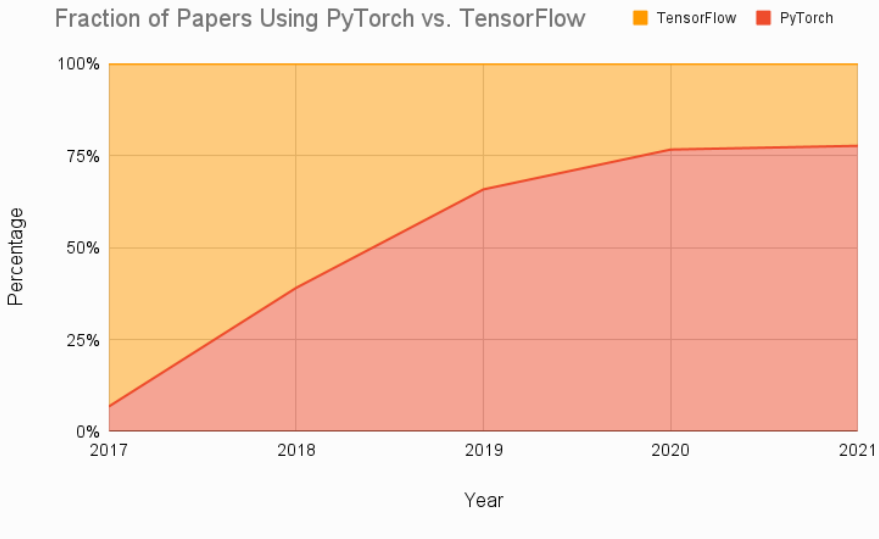
Introducción a la librería Pytorch

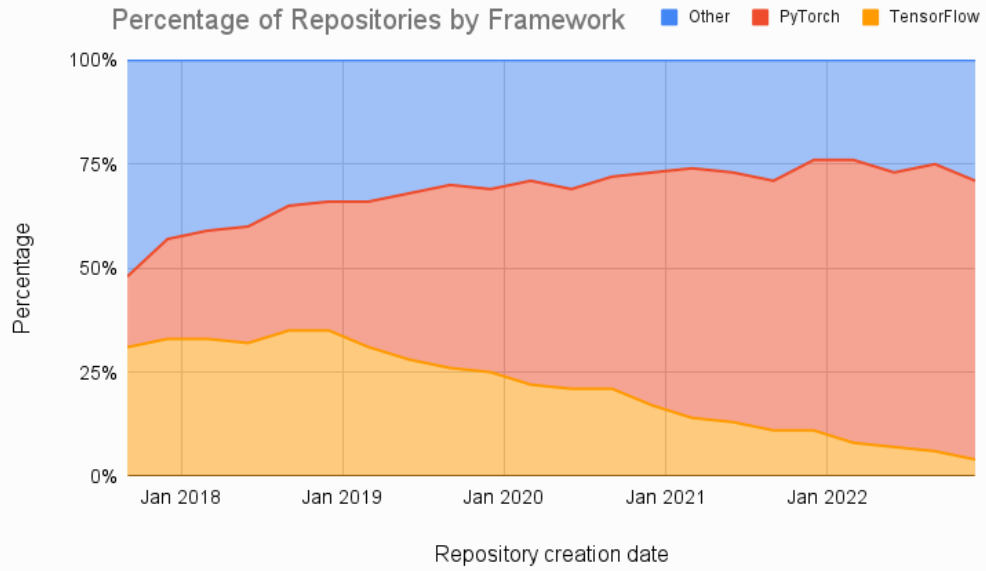
Diego Andrade Canosa

- Conceptos clave de Pytorch
  - Tensores
    - Transformaciones
  - Variables y gradientes
  - Diferenciación automática
- Creación y composición de la arquitectura de red
- 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

Introducción a la librería Pytorch


Paper Implementations grouped by framework
Other languages and frameworks
PyTorch
TensorFlow
JAX
MXNet
PaddlePaddle
torch
Caffe2
MindSpore
Jan 2023
PyTorch: 92% (141 repos)
TensorFlow: 8% (13 repos)
Share of Implementations
100%
75%
50%
25%
0%
Mar 18
Jun 18
Sep 18
Dec 18
Mar 19
Jun 19
Sep 19
Dec 19
Mar 20
Jun 20
Sep 20
Dec 20
Mar 21
Jun 21
Sep 21
Dec 21
Mar 22
Jun 22
Sep 22
Dec 22
Jan 23
Repository Creation Date


Fraction of Papers Using PyTorch vs. TensorFlow
TensorFlow
PyTorch
Percentage
100%
75%
50%
25%
0%
2017
2018
2019
2020
2021
Year


Percentage of Repositories by Framework
Other
PyTorch
TensorFlow
Percentage
100%
75%
50%
25%
0%
Jan 2018
Jan 2019
Jan 2020
Jan 2021
Jan 2022
Repository creation date

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

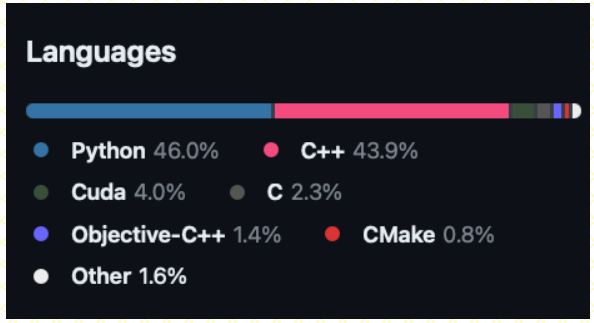
- Gran penetración en la comunidad investigadora
- Ventaja conceptual: Dynamic Computation Graph
- Facilidad de uso
- Flexibilidad
- Mucha base de código abierto en:
  - Visión por Computador
  - Procesado de Lenguaje Natural
  - Sistemas recomendadores
  - Procesado de señal

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

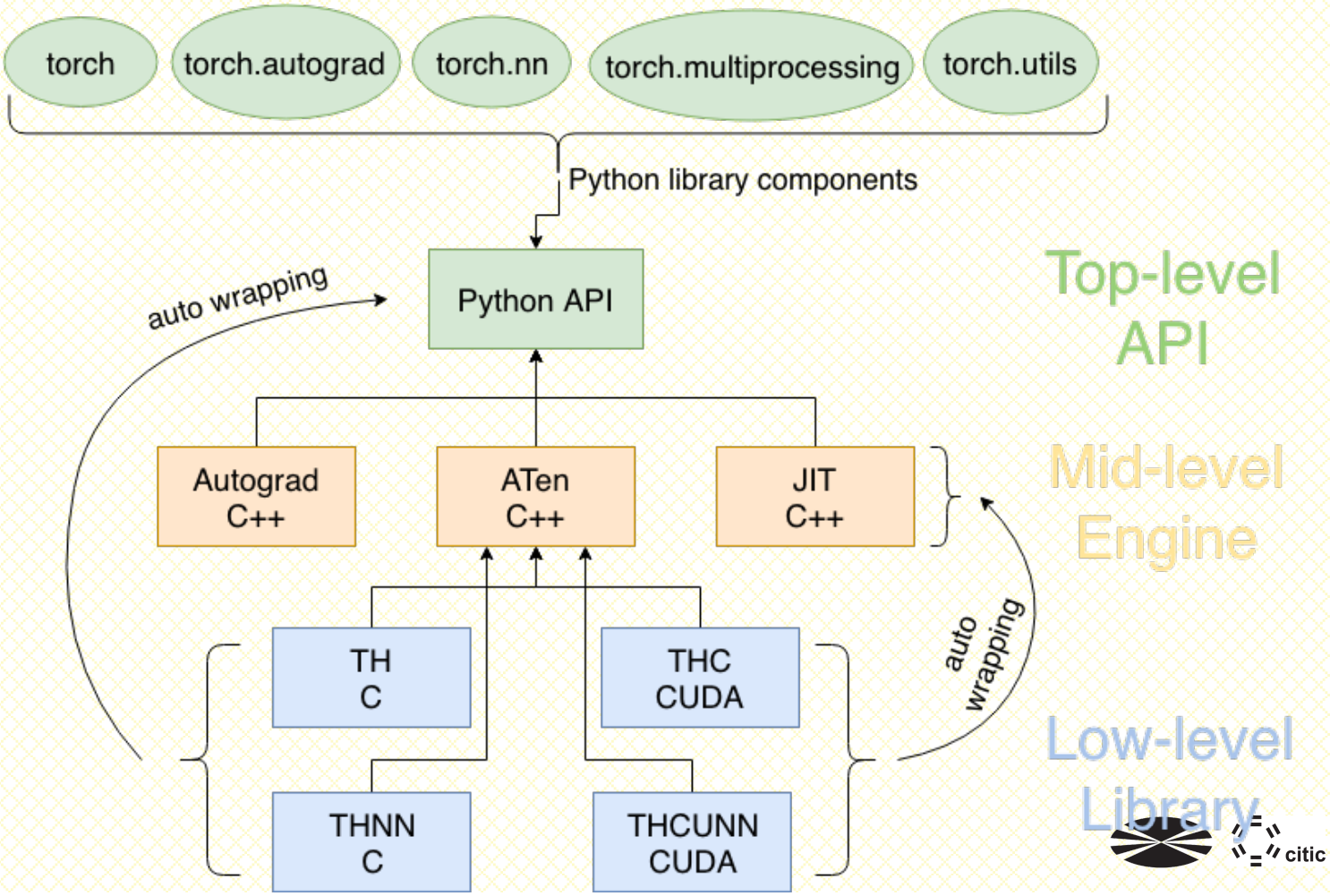
Principios de diseño

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

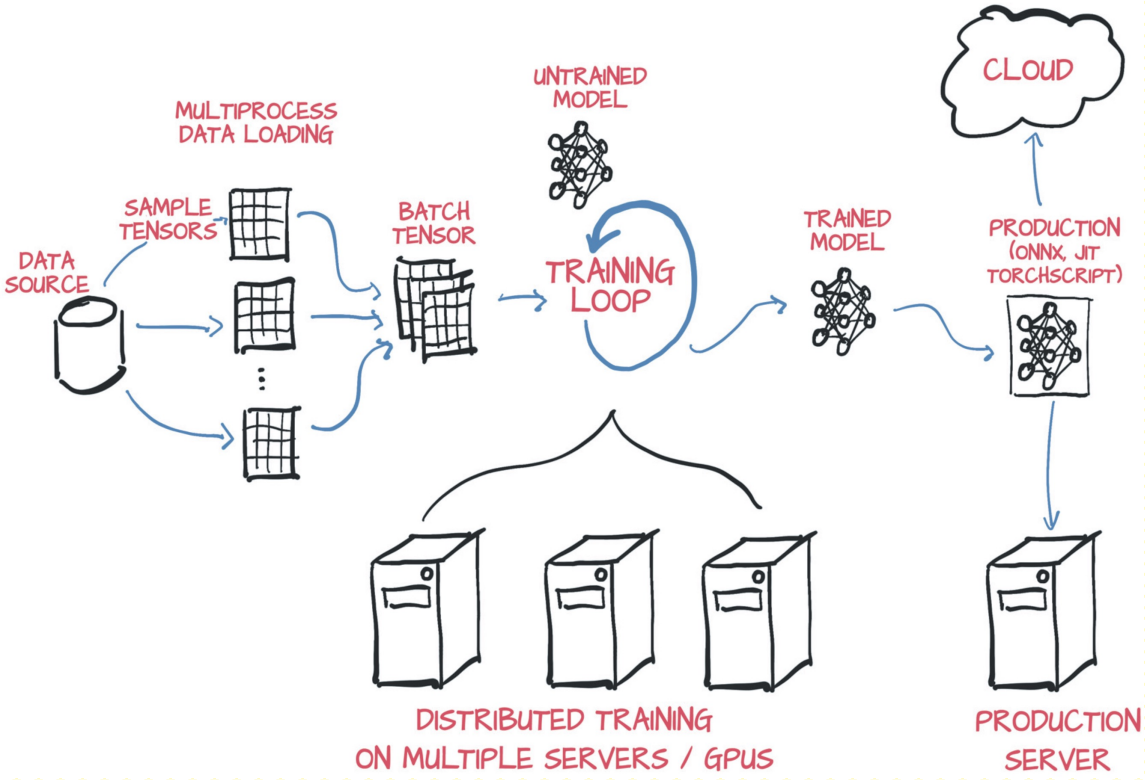


citic

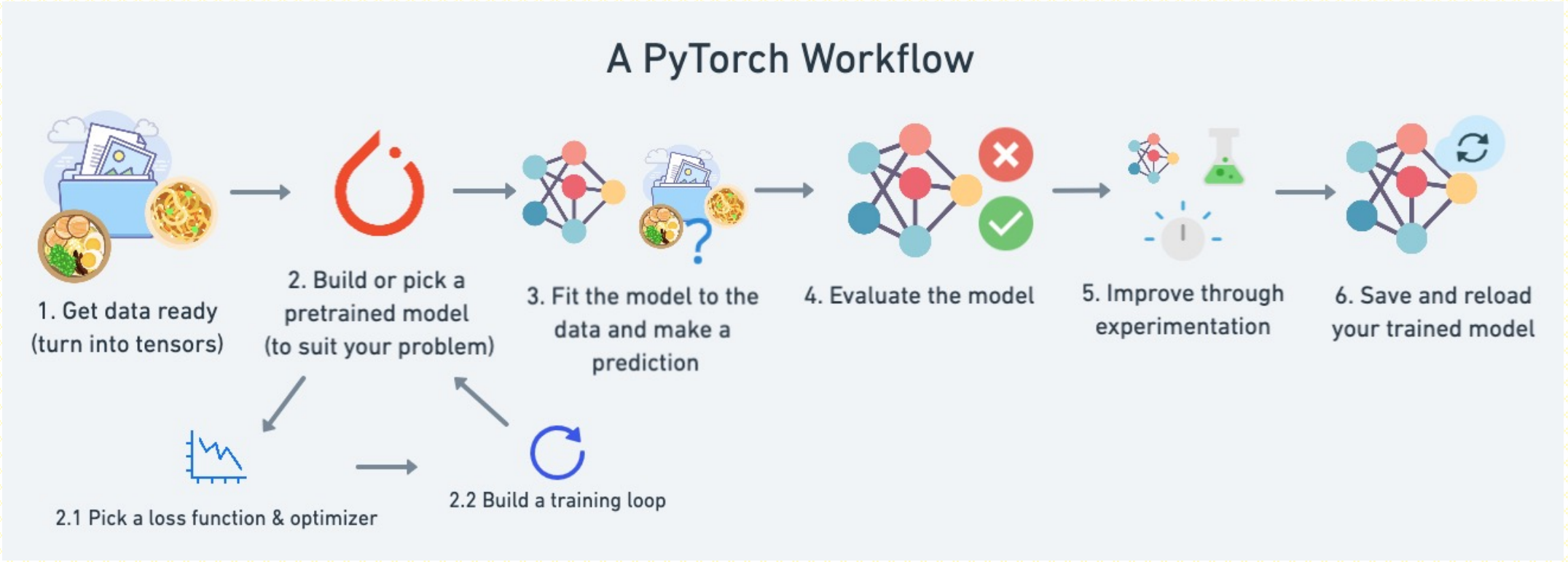
Estructura de módulos de Pytorch



Pytorch: Uso

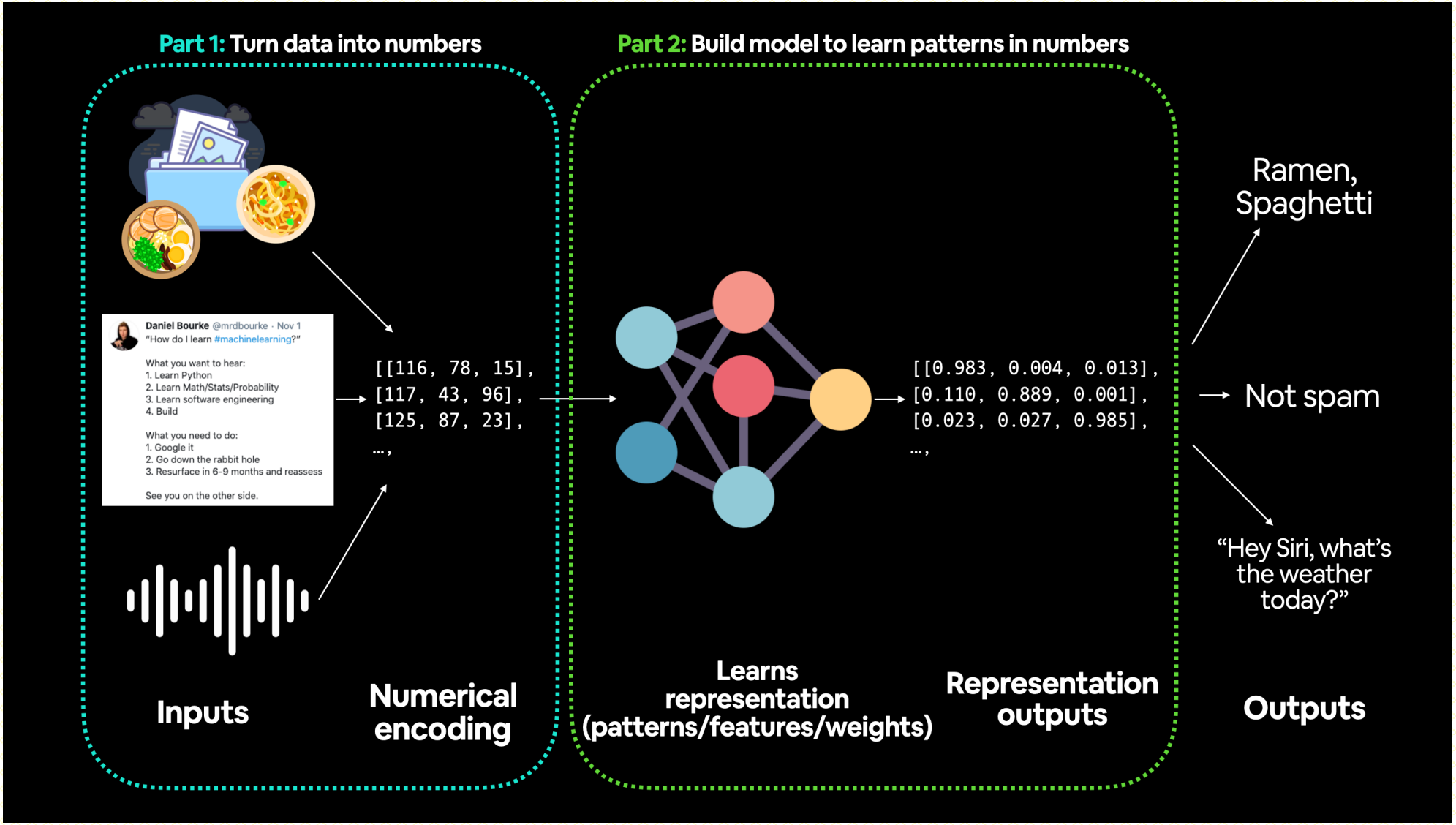


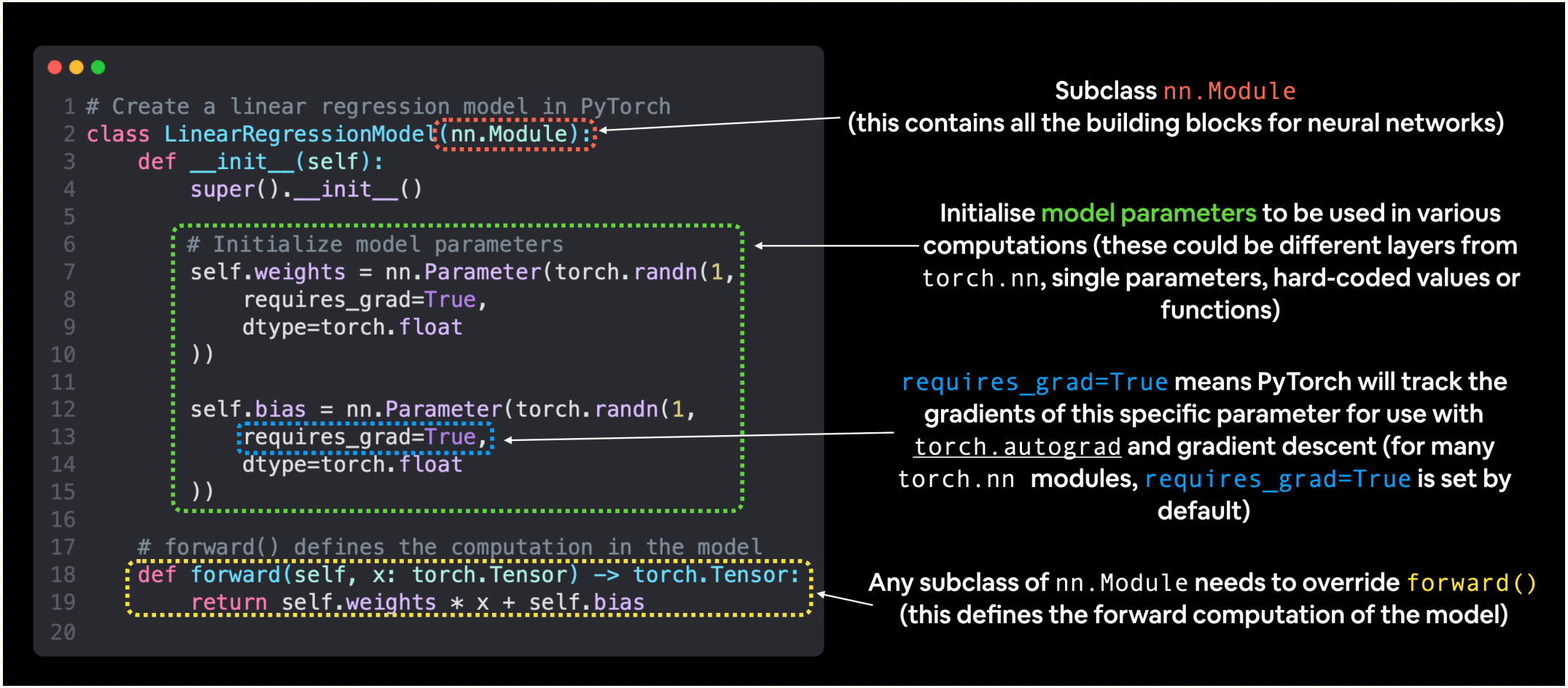
citic



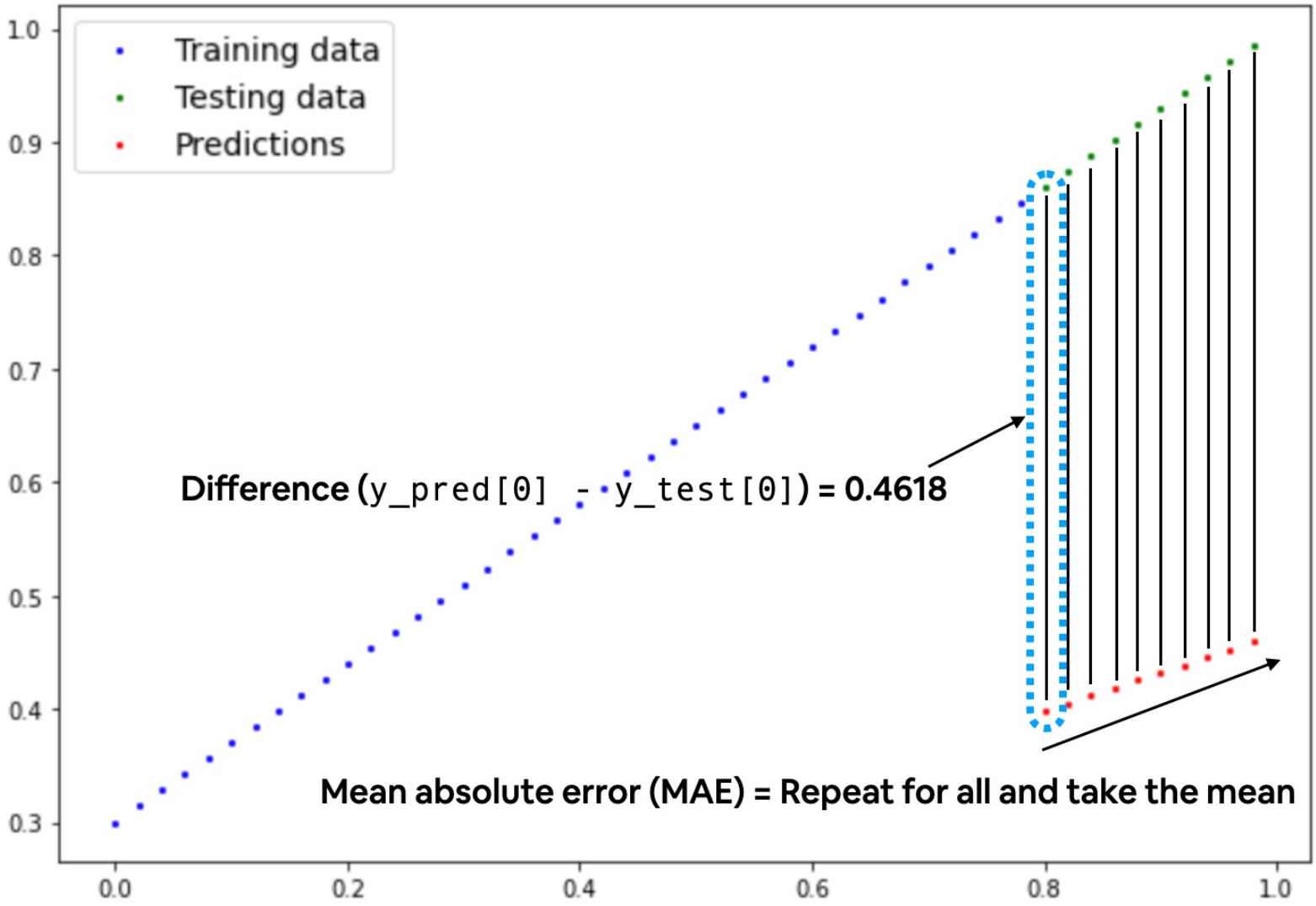
Fuente: https://www.learnpytorch.io/01_pytorch_workflow/

citic


Part 1: Turn data into numbers
Part 2: Build model to learn patterns in numbers
Daniel Bourke @mrdbourke · Nov 1
"How do I learn #machinelearning?"
What you want to hear:
1. Learn Python
2. Learn Math/Stats/Probability
3. Learn software engineering
4. Build
What you need to do:
1. Google it
2. Go down the rabbit hole
3. Resurface in 6-9 months and reassess
See you on the other side.
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
…,
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
…,
Ramen, Spaghetti
Not spam
"Hey Siri, what's the weather today?"
Inputs
Numerical encoding
Learns representation (patterns/features/weights)
Representation outputs
Outputs


1 # Create a linear regression model in PyTorch
2 class LinearRegressionModel(nn.Module):
3 def __init__(self):
4 super().__init__()
5
6 # Initialize model parameters
7 self.weights = nn.Parameter(torch.randn(1,
8 requires_grad=True,
9 dtype=torch.float
10))
11
12 self.bias = nn.Parameter(torch.randn(1,
13 requires_grad=True,
14 dtype=torch.float
15))
16
17 # forward() defines the computation in the model
18 def forward(self, x: torch.Tensor) -> torch.Tensor:
19 return self.weights * x + self.bias
20
Subclass nn.Module
(this contains all the building blocks for neural networks)
Initialise model parameters to be used in various computations (these could be different layers from torch.nn, single parameters, hard-coded values or functions)
requires_grad=True means PyTorch will track the gradients of this specific parameter for use with torch.autograd and gradient descent (for many torch.nn modules, requires_grad=True is set by default)
Any subclass of nn.Module needs to override forward()
(this defines the forward computation of the model)

citic


Training data
Testing data
Predictions
Difference (y_pred[0] - y_test[0]) = 0.4618
Mean absolute error (MAE) = Repeat for all and take the mean
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
0.2
0.4
0.6
0.8
1.0

citic

PyTorch training loop

```python
# Pass the data through the model for a number of epochs (e.g. 100)
for epoch in range(epochs):

    # Put model in training mode (this is the default state of a model)
    model.train()

    # 1. Forward pass on train data using the forward() method inside
    y_pred = model(X_train)

    # 2. Calculate the loss (how different are the model's predictions to the true values)
    loss = loss_fn(y_pred, y_true)

    # 3. Zero the gradients of the optimizer (they accumulate by default)
    optimizer.zero_grad()

    # 4. Perform backpropagation on the loss
    loss.backward()

    # 5. Progress/step the optimizer (gradient descent)
    optimizer.step()
```

Note: all of this can be turned into a function

Pass the data through the model for a number of epochs (e.g. 100 for 100 passes of the data)

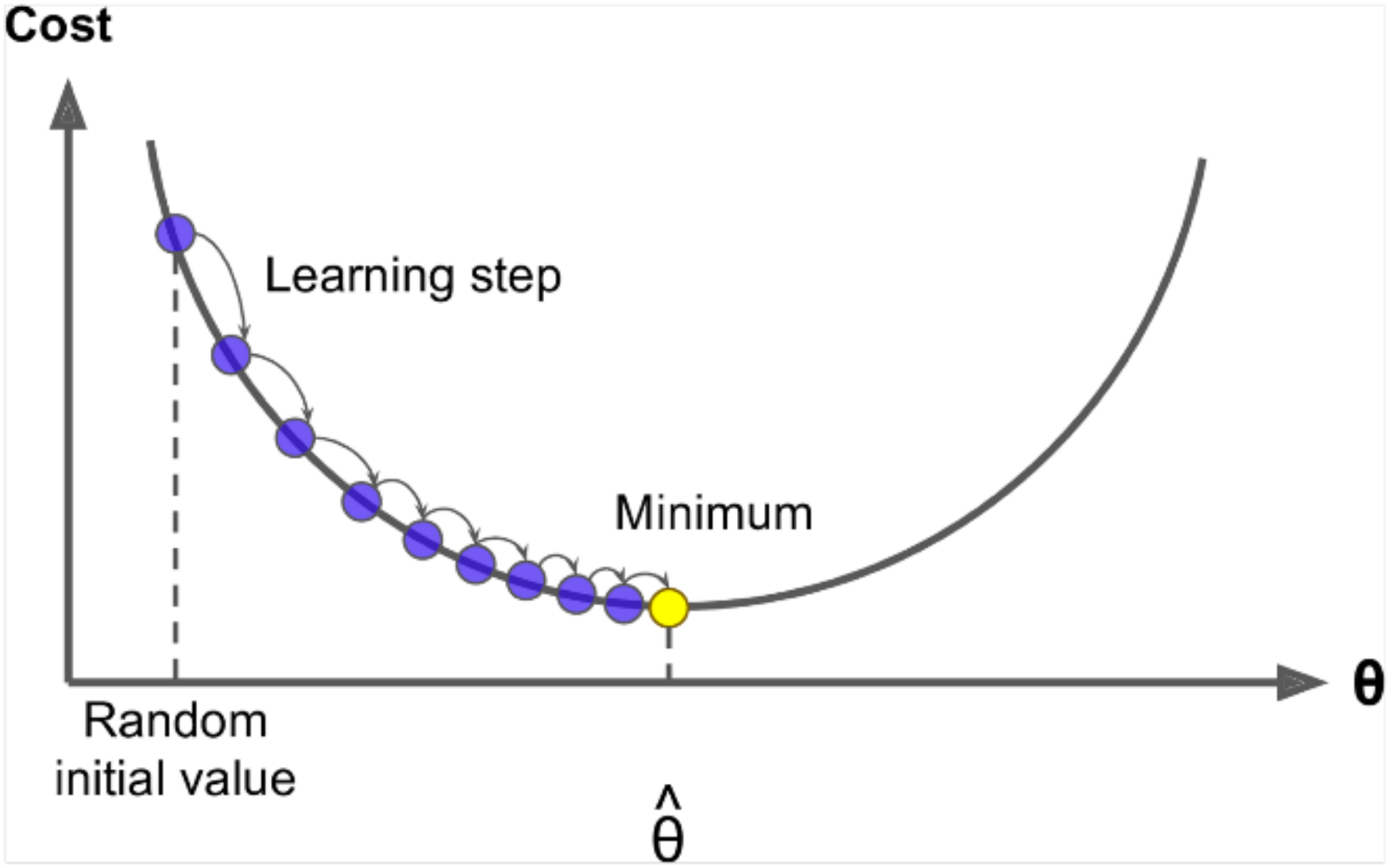
Pass the data through the model, this will perform the `forward()` method located within the model object

Calculate the loss value (how wrong the model's predictions are)

Zero the optimizer gradients (they accumulate every epoch, zero them to start fresh each forward pass)

Perform backpropagation on the loss function (compute the gradient of every parameter with `requires_grad=True`)

Step the optimizer to update the model's parameters with respect to the gradients calculated by `loss.backward()`


Cost
Learning step
Minimum
θ
Random
initial value
θ̂

PyTorch testing loop

```python
# Setup empty lists to keep track of model progress
epoch_count = []
train_loss_values = []
test_loss_values = []

# Pass the data through the model for a number of epochs (e.g. 100) pochs):
for epoch in range(epochs):

    ### Training loop code here ###

    ### Testing starts ###

    # Put the model in evaluation mode
    model.eval()

    # Turn on inference mode context manager
    with torch.inference_mode():
      # 1. Forward pass on test data
      test_pred = model(X_test)

      # 2. Caculate loss on test data
      test_loss = loss_fn(test_pred, y_test)

    # Print out what's happening every 10 epochs
    if epoch % 10 == 0:
        epoch_count.append(epoch)
        train_loss_values.append(loss)
        test_loss_values.append(test_loss)
        print(f"Epoch: {epoch} | MAE Train Loss: {loss} | MAE Test Loss: {test_loss} ")
```

Note: all of this can be turned into a function

Create empty lists for storing useful values (helpful for tracking model progress)

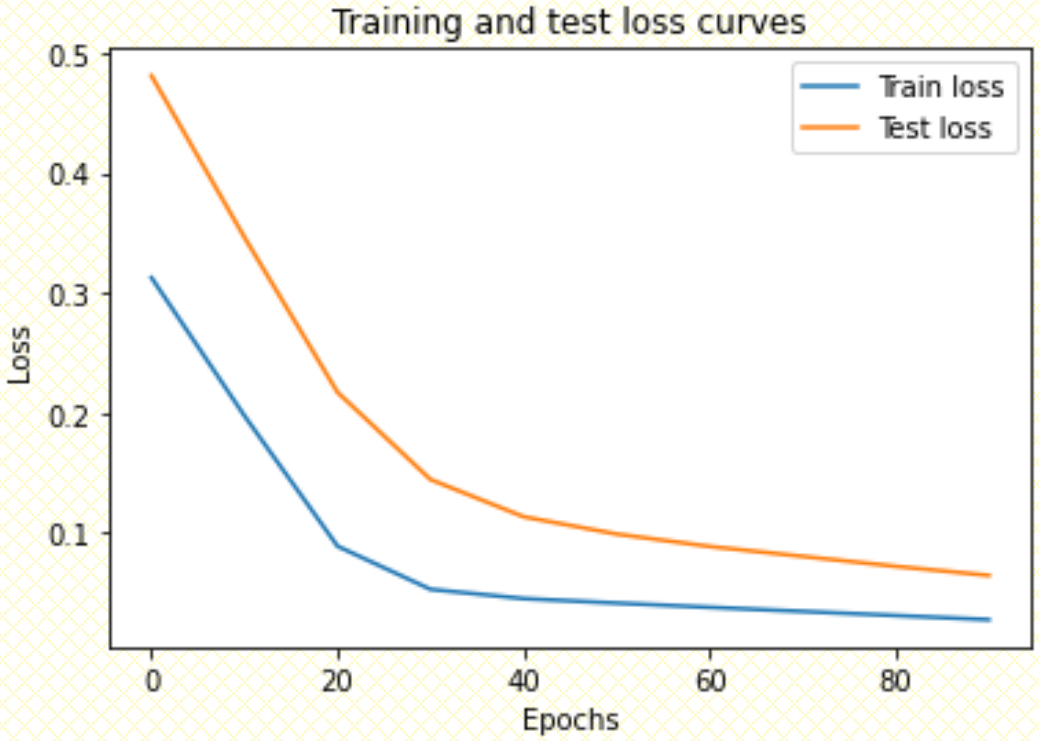
Tell the model we want to evaluate rather than train (this turns off functionality used for training but not evaluation)

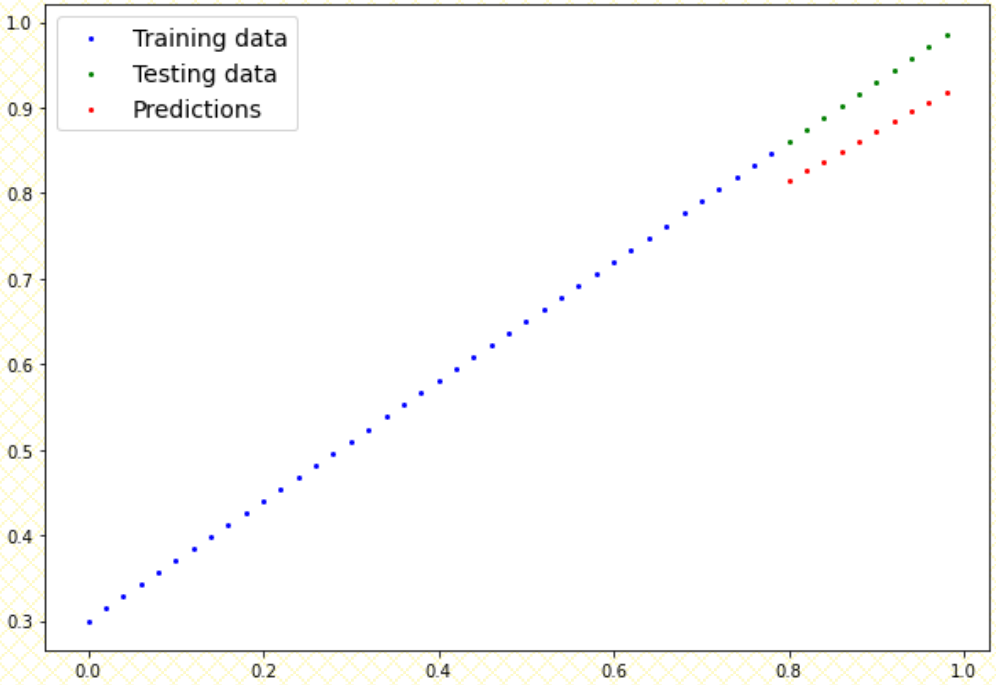
Turn on `torch.inference_mode()` context manager to disable functionality such as gradient tracking for inference (gradient tracking not needed for inference)

Pass the test data through the model (this will call the model's implemented `forward()` method)

Calculate the test loss value (how wrong the model's predictions are on the test dataset, lower is better)

Display information outputs for how the model is doing during training/testing every ~10 epochs (note: what gets printed out here can be adjusted for specific problems)

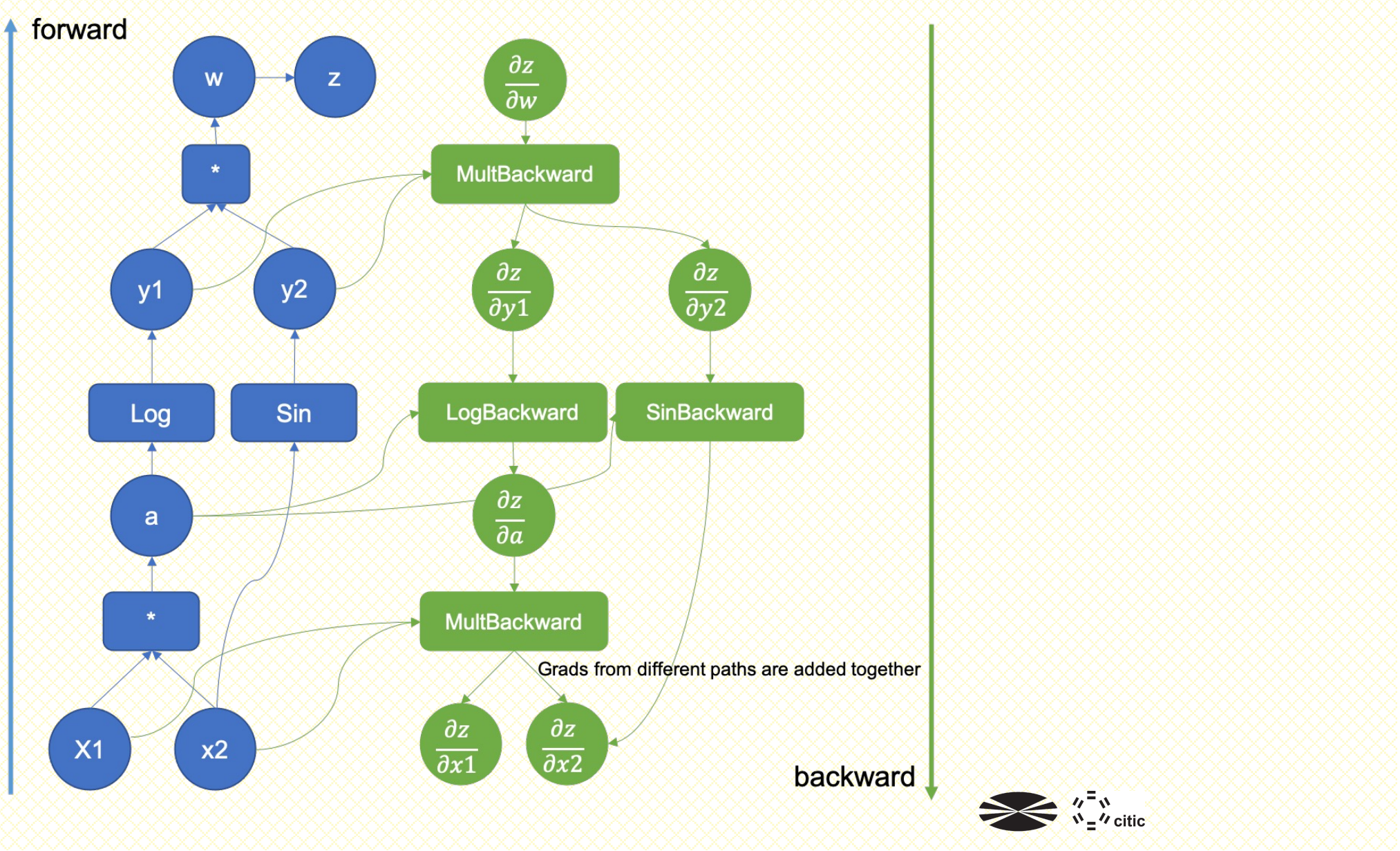

Training and test loss curves
Train loss
Test loss
Loss
Epochs
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80


Training data
Testing data
Predictions
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
0.2
0.4
0.6
0.8
1.0

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

Abstracciones

- Tensores: estructura de datos multidimensional donde se almacenan los diversos elementos del modelo: parámetros (pesos), bias, entradas, salidas
- Contenedores (*container*): 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

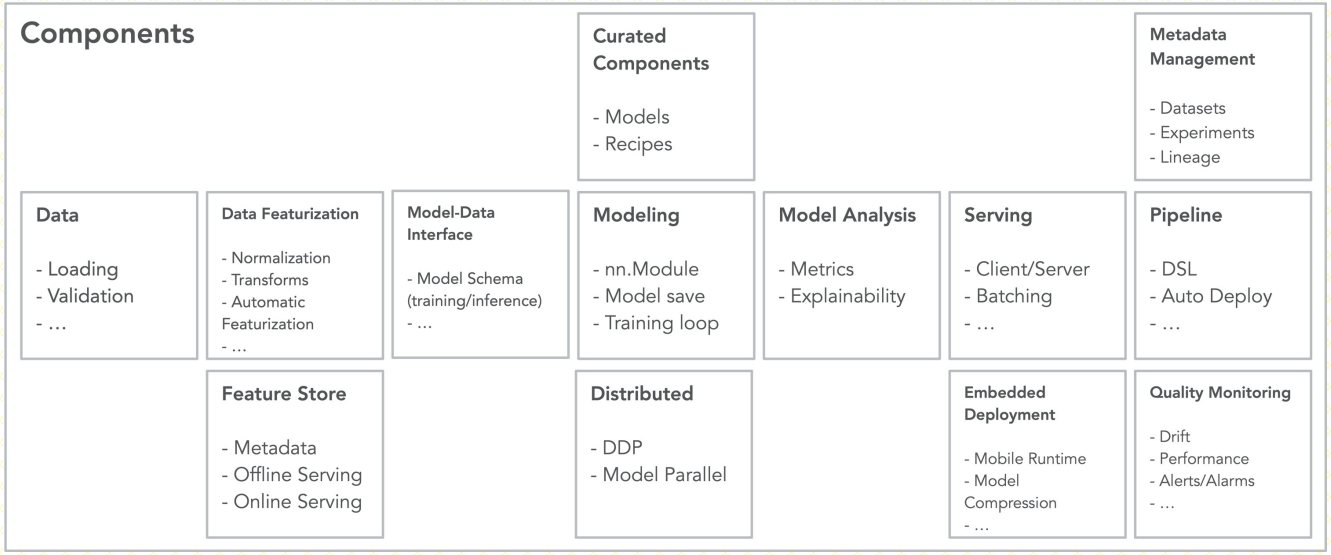
Lanzador para Pytorch

Tipo srun o qsub

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

Permite ejecución local, remota y distribuida

https://pytorch.org/torchx/latest/quickstart.html



citic

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/